А. А. КЛЮЧАРЁВ
А. А. ФОМЕНКОВА

ТИПЫ И СТРУКТУРЫ ДАННЫХ В ИНФОРМАТИКЕ И ПРОГРАММИРОВАНИИ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

А. А. Ключарёв, А. А. Фоменкова

ТИПЫ И СТРУКТУРЫ ДАННЫХ В ИНФОРМАТИКЕ И ПРОГРАММИРОВАНИИ

Учебное пособие

ГУАП
Санкт-Петербург
2021

УДК 004.421.6(075)
ББК 32.972.134я73
К52

Рецензенты:
доктор педагогических наук, профессор *А. Г. Степанов*;
доктор технических наук, профессор *В. В. Шмелев*

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

Протокол № 6 от 1 ноября 2020 г.

Ключарёв, А. А.

К52 Типы и структуры данных в информатике и программировании: учеб. пособие / А. А. Ключарёв, А. А. Фоменкова. – СПб.: ГУАП, 2021. – 103 с.

ISBN 978-5-8088-1561-2

Приводится подробное описание простейших типов данных и основных алгоритмов их обработки применительно к различному восприятию данных в процессах разработки и функционирования информационных систем.

Предназначено для изучения теоретического материала по дисциплине «Информатика» студентами, обучающимися по направлениям 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.04 – «Программная инженерия» и 01.03.02 –«Прикладная математика и информатика», и может быть использовано при изучении аналогичной дисциплины студентами инженерных направлений.

УДК 004.421.6(075)
ББК 32.972.134я73

ISBN 978-5-8088-1561-2

ВВЕДЕНИЕ

Учебное пособие предназначено, в первую очередь, для студентов, обучающихся по направлениям, связанным с использованием информационных технологий и программированием, и содержит материал по существу являющийся вводным при изучении специальных дисциплин. В дисциплине «Информатика» основными понятиями, требующими детального рассмотрения, являются «информация» и «данные».

Логически пособие состоит из трех частей.

В первых двух разделах рассматриваются основные теоретические положения, лежащие в основе общего подхода к рассмотрению структуры информационных процессов, и вводятся основные определения понятий, используемых в них. Рассматривается вычислительная машина как основное устройство обработки данных.

Основной материал изложен в разделах 3–10 и посвящен рассмотрению простейших типов данных и структур данных с позиций их восприятия специалистами различного уровня и конечным пользователем. Особое внимание уделено теоретическим аспектам, не зависящим от используемых технологий реализации информационной системы.

Третья часть пособия, представленная в разделах 11–13, иллюстрирует основные теоретические положения на примере их реализации в языке программирования C/C++, изучаемом студентами в дисциплине «Основы программирования», что на взгляд авторов должно способствовать логической связи дисциплин и лучшему пониманию студентами особенностей обработки данных.

1. ИНФОРМАЦИЯ И ДАННЫЕ В ИНФОРМАЦИОННЫХ ПРОЦЕССАХ

Рассматриваемые в настоящем учебном пособии типы и структуры данных являются разделом дисциплины «Информатика» и, естественно, необходимо определить ее предметную область и содержание.

За основу возьмем определение, приведенное в Большой российской энциклопедии в 2008 г. [1]:

«**Информатика** (англ. informatics) – наука о методах и процессах сбора, хранения, обработки, передачи, анализа и оценки информации с применением компьютерных технологий, обеспечивающих возможность её использования для принятия решений».

Это краткое определение расширено в последней редакции энциклопедии, имеется ряд трактовок термина в различных публикациях. Считается, что синонимом слова «Информатика» является понятие «Computer science», принятое в англоязычной литературе, но оно не подчеркивает именно фундаментальный характер информатики как науки [1].

Как показано на рис. 1, слово «информатика» образовано из двух слов «информация» и «автоматика».

Под «автоматикой» в данном контексте подразумеваются средства сбора, передачи, хранения и обработки информации, которые, в свою очередь, можно разделить на технические средства (Hardware) и программное обеспечение, реализующее заданные алгоритмы обработки информации (Software). Такое разделение обусловлено тем, что основным средством, решающим задачи информатики, являются вычислительные устройства и системы.

Понятие «информация» происходит от латинского слова informatio – разъяснение, осведомление, изложение и обозначает одно

Рис. 1. Структура дисциплины «Информатика»

из основных свойств материи. Термин «информация» является первичным и не имеет конкретного определения. Его смысл различен в зависимости от конкретной предметной области, в которой он используется, и решаемыми задачами. Например, термин «информация» имеет различное значение в журналистике, медицине, технике и в других областях. Поэтому в литературе можно встретить множество определений этого понятия. Часто эти определения сложны и противоречивы, поэтому следует определить понятие «информация», используемое нами в рамках данного курса.

Применительно к задачам, решаемым в области информационных технологий, будем полагать, что

Информация – любые сведения об объекте окружающего мира, являющиеся объектом хранения, передачи, преобразования и обработки [2].

В основополагающей статье К. Шеннона [3] сформулировано, что понятие информации складывается из трех аспектов – семантического, прагматического и синтаксического. Семантический подход изучает смысловое содержание информации, отражающее отношение между формой сообщения в конкретной языковой среде и его смысловым содержанием. Прагматический подход основан на рассмотрении информации с точки зрения полезности для решения конкретной задачи. Эти подходы имеют существенное значение в лингвистике, системах искусственного интеллекта и других областях, а изучение их свойств, количественная оценка и другие аспекты являются предметом отдельной науки – теории информации.

В информатике, рассматриваемой в соответствии с приведенным выше определением, чаще используется синтаксический подход, в котором смысловое значение информации не учитывается, а предметом изучения являются ее характеристики, определяющие процессы хранения, передачи и алгоритмы обработки. Именно этот подход определяет содержание дисциплины «Информатика» для технических специальностей, что часто приводит к идентичности понятий «информация» и «данные», если это не противоречит смыслу решаемых задач.

В любом случае в информационном процессе всегда можно выделить три компоненты (см. рис. 2) – источник информации, получатель информации и канал связи.

На основе рис. 2. можно выделить некоторые интуитивно понятные свойства информации:

– информация переносит знания об объекте, которых в рассматриваемой точке не было до получения информации. Если у полу-

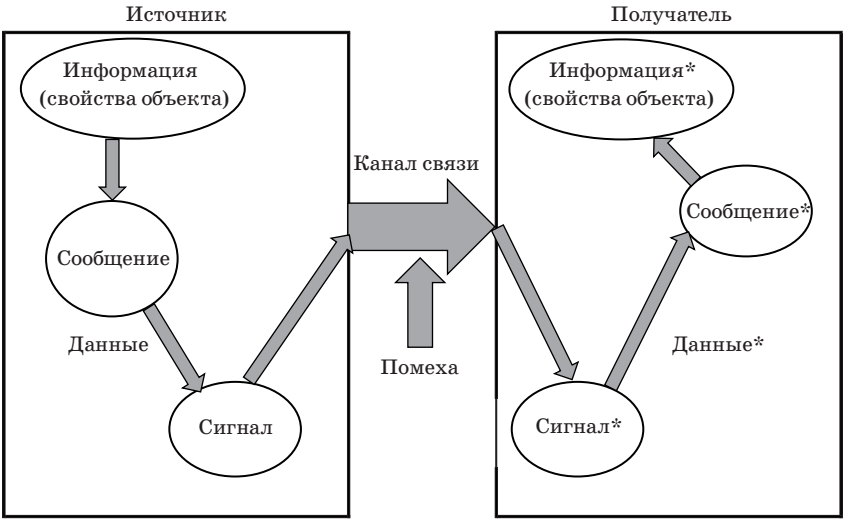


Рис. 2. Основные составляющие информационного процесса

чателя информации уже есть определенные сведения об объекте, то для него эти сведения уже не являются информацией, хотя их можно передавать произвольное число раз, но это уже будут просто данные, которые не несут информации получателю;

– информация не материальна и, следовательно, ее нельзя передать – переместить от источника к получателю информации. Для передачи информации требуется некоторый материальный носитель, непосредственно используемый в канале связи и некоторым образом связанный с передаваемой информацией;

– информация может быть заключена в символах (знаках) или их последовательности в определенном порядке. В качестве символов могут использоваться элементы различных языков общения – символы определенного национального алфавита, графические изображения, например, дорожные знаки, графические или звуковые образы. Эта последовательность называется ***сообщение***. Сообщение, необходимое для решения некоторой задачи или совокупности задач, представленное в форме, обеспечивающей его распознавание, называют ***данные***. На различных этапах информационного процесса форма представления данных различна. Так, например, человек воспринимает данные в символьной форме, а аппаратные средства и программное обеспечение вычислительного устройства в виде закодированной последовательности двоичных чисел. Важно отметить,

что сообщение так же не материально и для его передачи требуется материальный носитель;

– сообщение может быть передано по каналу связи только с помощью материального носителя. Таким является ***сигнал*** – изменение энергии, однозначно связанное с передаваемыми символами, составляющими сообщение;

– символы и сигналы несут информацию только для получателя, который может их распознать, т. е. правила преобразований, отраженных на рис. 1, должны быть согласованы, обычно это реализуется определенными способами кодирования и декодирования.

На различных этапах информационного процесса выполняется преобразование формы представления данных. Обычно, правила и алгоритмы преобразования называют к**одирование**, под которым подразумевают процесс преобразования данных из формы, необходимой для непосредственного использования информации, в форму, воспринимаемую устройствами передачи, хранения и программным обеспечением, осуществляющим обработку данных. В реальных системах выполняется многократное кодирование, на каждом этапе которого решается ряд специфических задач, обеспечивающих требуемое качество функционирования системы. Процессы обратного преобразования называют **декодировани**ем. Очевидно, что правила кодирования и декодирования в системе должны быть согласованы.

В любом канале связи на передаваемый сигнал воздействует помеха. В результате на входе получателя присутствует смесь полезного сигнала и помехи и, следовательно, принятый сигнал никогда не будет точной копией переданного. На рис. 1 этот факт отмечен символом «*». Таким образом, в результате декодирования выделенное из сигнала сообщение всегда будет отличаться от переданного, что приведет, в конечном итоге, к ошибкам в интерпретации сообщения при получении информации. Одним из разделов информатики как фундаментальной науки является разработка методов уменьшения влияния ошибок и, следовательно, повышения достоверности получаемой информации [1,2].

Таким образом, любой информационный процесс, может быть представлен как процесс передачи информации от объекта, являющегося источником информации, к получателю. Для обеспечения передачи информации необходим канал связи. Это некоторая физическая среда, через которую информация, заключенная в сообщении с помощью изменения энергии сигнала, передается получателю. Например, в цифровых системах информация передается в виде

последовательности двоичных чисел представленных символами «0» или «1» и в канале связи каждый символ кодируется определенным уровнем энергии. При использовании проводного канала связи это может быть наличие или отсутствие тока в электрической цепи.

Множество всех символов a_1, a_2, ..., a_k, использующееся для формирования сообщения, называется алфавит

$$A = \{a_1, a_2, ..., a_k\}.$$

Обычно конкретное изображение символа имеет значение только при использовании конкретной языковой среды. При формировании сообщения важно не изображение символа, а количество символов, используемых в системе. Поэтому чаще используется понятие размер (мощность) алфавита, которое равно количеству символов в алфавите и обычно также обозначается «A». Знаками алфавита размером A может быть передано N = A сообщений.

Из знаков алфавита может быть составлено слово. Если размер слова фиксирован и составляет n знаков, то количество возможных слов N составленных из символов алфавита A таким образом, что каждый символ алфавита может входить в слово 0, 1, 2, ..., n, раз определяется

$$N = A^n. \quad (1)$$

Таким образом, с помощью слов можно представить информацию о любом из N сообщений.

Выражение (1.1) позволяет определить размер слова из алфавита A, с помощью которого можно представить N сообщений

$$n = \lceil \log_A N \rceil. \quad (2)$$

Мы можем сопоставить тому или иному сообщению комбинацию знаков, тогда при приеме сообщения, зная правила сопоставления, можно распознать сообщение.

Рассмотренная модель информационного процесса была предложена в работе [3] и в настоящее время является общепринятой. Современные информационные системы, как правило, используют компьютерные технологии. Поэтому рассматривать вопросы обработки данных в этих системах следует с учетом особенностей их представления в вычислительных машинах.

2. СТРУКТУРА И ПРИНЦИП ДЕЙСТВИЯ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ. ПРИНЦИПЫ ФОН НЕЙМАНА

Большинство широко используемых современных вычислительных машин относятся к классу машин «фон Неймановского типа». Несмотря на то, что используемые сегодня вычислительные устройства существенно отличаются от первых вычислительных машин, основополагающие принципы их работы практически не изменились. Поэтому считается, что эти компьютеры имеют архитектуру фон Неймана, подразумевая, что в машине используется принципы программного управления и совместного хранения программ и данных в памяти. Основные принципы построения вычислительных машин с фон Неймановской архитектурой были опубликованы в классической статье Фон Неймана, Голдстайна и Бёркса «Предварительное рассмотрение логической конструкции электронно-вычислительного устройства» в 1946 г. и до сегодняшнего дня определяют особенности использование компьютерных технологий. Структура вычислительной машины с программным управлением приведена на рис. 3.

Учитывая, что изучение основных принципов построения вычислительной машины с программным управлением (принципов фон Неймана) входит в программу дисциплины «Информатика» средней школы, здесь мы только кратко приведем их перечисление, принятое в современных учебниках [1, 2]:

– двоичное представление команд и данных;
– программное управление;
– адресуемость памяти;
– однородность памяти;
– возможность перехода в процессе выполнения программы.

Рассмотрим более подробно содержание перечисленных принципов применительно особенностям организации и обработки данных.

Принцип двоичного представления данных обосновывает представление любых объектов в виде двоичных чилел. Доказано, что именно использование двоичной системы счисления позволяет минимизировать аппаратные затраты и обеспечить эффективность выполнения арифметических и логических операций. Таким образом, вычислительная машина является автоматом, выполняющим преобразование двоичных чисел по заданному алгоритму.

Любая вычислительная машина состоит из двух основных блоков – процессора и памяти (рис. 3). Устройства ввода-вывода, указанные на рисунке, необходимы для взаимодействия машины с внешней средой и не являются определяющими в принципах ее работы. В памяти хранятся только двоичные числа. Память вычислительной машины логически разделена на две области, которые содержат программы и данные, представленные в виде последовательности двоичных чисел. Эти области определяются не местом их размещения в памяти, а способом использования содержимого ячеек памяти. Одно и то же двоичное число, хранимое в памяти машины, может использоваться либо как элемент данных, либо как команда процессору, задающая преобразование данных в программе. В результате обеспечивается возможность использования содержимого области программ как данные и наоборот. Такой подход принципиально обеспечивает возможность описания алгоритма на формализованном языке высокого уровня, автоматического преобразования текста программы в последовательность двоичных чисел, пред-

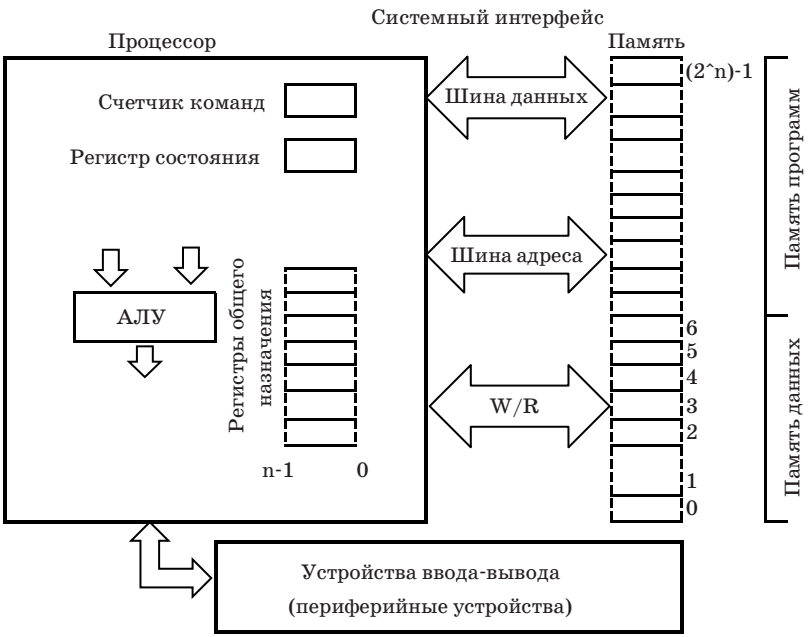


Рис. 3. Обобщенная структура вычислительной машины с программным управлением

ставляющих программу в памяти, и выполнение этой программы. Таким образом, архитектура фон Неймана обеспечивает взаимодействие человека с вычислительной машиной на уровне языков программирования, позволяющих описывать алгоритмы обработки данных в естественной для человека форме, а не в виде последовательности двоичных чисел.

Процессор представляет собой устройство, которое выполняет преобразование данных, хранящихся в памяти, под управлением программы, так же хранимой в памяти. Счетчик команд обеспечивают последовательную выборку команд в порядке их следования в памяти и их выполнение процессором. В счетчик команд хранится адрес команды, подлежащей выполнению. При чтении очередной команды из памяти счетчик команд автоматически увеличивается на длину этой команды и, следовательно, в нем образуется адрес следующей (по расположению в памяти) команды. Таким образом, обеспечивается последовательное выполнение команд, размещенных в памяти. Изменение этого порядка, например, при вызове функций, выполнении циклов или условных переходов обеспечивается специальными командами ветвления, которые дают возможность перехода в процессе выполнения программы к другому фрагменту программного кода.

В современных машинах процессор имеет регистры общего назначения, предназначенные для временного хранения данных, результатов операций над данными, адресов и некоторых специальных данных, используемых конкретной вычислительной системой.

Основным блоком процессора является арифметико-логическое устройство (АЛУ), которое может выполнять арифметические и логические операции над данными. Исходные данные (операнды) временно хранятся в регистрах процессора. Результат вычислений так же помещается в один из регистров. Как правило, АЛУ позволяет выполнять только операции сложения целых чисел, несколько логических операций, образующих функционально полный набор логических функций (дизъюнкция, конъюнкция, отрицание, сумма по модулю 2) и операции сдвига.

Вспомогательные функции выполняет регистр состояния процессора, в котором формируются признаки выполнения отдельных операций по преобразованию данных, например, равенство результата нулю, знак результата, переполнение и другие. Эти признаки используются для управления вычислительным процессом использованием команд ветвления, нарушающих последовательную выборку команд из памяти при получении определенного результата вычислений.

С точки зрения обработки данных значение имеют принципы адресуемости и однородности памяти.

Память состоит из ячеек одинакового размера. Любое данное может занимать в памяти только целое количество ячеек. Все ячейки памяти имеют уникальный номер, называемый адресом. Память линейна и однородна.

Говоря о линейности памяти, имеют в виду, что все ячейки имеют адреса с последовательно нарастающим номером. Если процессор оперирует двоичными словами разрядностью **n**, то, в соответствии с (1), в вычислительной машине существует $\mathbf{2^n - 1}$ ячеек памяти, адреса которых задаются двоичными числами от **0** до $\mathbf{2^n - 1}$.

Принцип однородности памяти подразумевает, что любое двоичное число, хранимое в памяти, может обозначать совершенно произвольный объект. Логически это может быть число, символ, команда для процессора или что нибудь другое. Все определяется только тем, как и по каким правилам это число используется. При программировании на уровне машинных команд или на языке ASSEMBLER эти правила устанавливает программист, реализуя заданный алгоритм работы программы. В высокоуровневых системах программирования используются определенные соглашения о правилах интерпретации содержимого ячеек памяти и автоматический контроль их соблюдения. Определяющим понятием в реализации этих правил является **тип данных**.

Поясним изложенное выше на простом примере сложения двух чисел. В записи на некотором языке программирования этому может соответствовать инструкция в виде

```
var_c = var_a + var_b;
```

которая означает «переменной `var_c` присвоить значение `var_a + var_b`». Благодаря рассмотренным особенностям архитектуры фон Неймана такая строка символов может быть автоматически переведена в машинную команду, каждая компонента которой есть двоичные числа, обозначающие различные объекты – действие, адреса операндов, адрес результата

«сложить» «адрес `var_a`» «адрес `var_b`» «адрес `var_c`».

Перечисленные особенности работы вычислительной машины важно понимать при использовании различных данных в вычислительных системах.

3. ПРЕДСТАВЛЕНИЕ ДАННЫХ В КОМПЬЮТЕРНЫХ СИСТЕМАХ

3.1. Уровни представления данных в компьютерных системах

Для уточнения подходов к представлению данных в компьютерных системах рассмотрим упрощенную схему взаимодействия пользователя с вычислительной машиной (рис. 4).

Компьютерная система в общем случае предназначена для предоставления данных конечному пользователю, который имеет возможность извлечения знаний из потока данных (сообщений), формируемых системой. Часто в качестве конечного пользователя выступает человек, являющийся специалистом в конкретной предметной области, т. е. знающий правила интерпретации потока данных в конкретную информацию об объекте. Эта информация обычно используется конечным пользователем для принятия решения о свойствах объекта либо управлением его состоянием. Отметим, что в качестве конечного пользователя может выступать не только человек, но и некоторая техническая система, например система искусственного интеллекта или система автоматического управления объектом. Здесь важным является наличие возможности и знание правил извлечения информации из данных. Конечный пользователь представляет данные в терминах своей предметной области. Эти данные, как правило, недостаточно формализованы для дальнейшей обработки в вычислительной системе, работающей

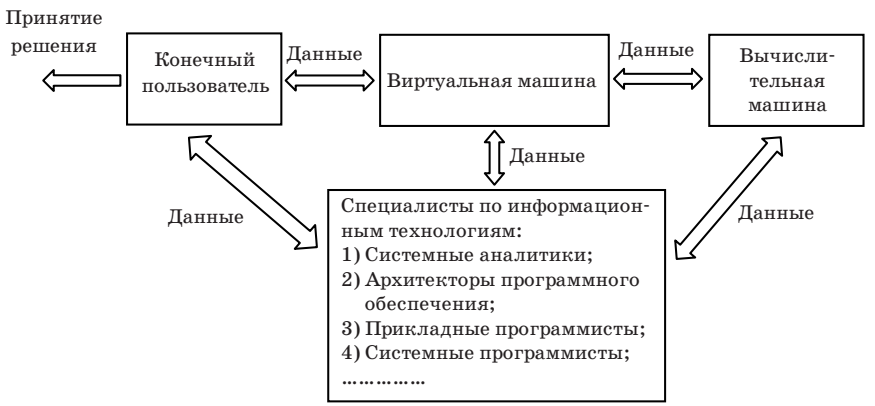


Рис. 4. Уровни представления данных в компьютерных системах

по жестким алгоритмам и, тем более, непригодны для использования в вычислительной машине, воспринимающей только двоичные числа.

Общепринятым в настоящее время является подход, заключающийся в предоставлении конечному пользователю некоторой виртуальной вычислительной машины, предоставляющей пользователю естественный интерфейс для общения с вычислительной системой. Задачами создания и обеспечения функционирования этой виртуальной машины занимаются специалисты в области информационных технологий. Эти задачи имеют множество специфических аспектов, и их решение требует, с одной стороны, максимального учета требований конечного пользователя, а с другой эффективного использования ресурсов реальной вычислительной системы для сбора, хранения, обработки, передачи и анализа данных и, в конечном итоге, направлены на создание программного обеспечения. Очевидно, что это не только задачи программирования. В этом процессе участвуют:

– **Системный аналитик**, в информационных технологиях отвечает за анализ потребностей конечного пользователя с целью их выполнения при реализации системы. В работе системного аналитика используются методы системного анализа и математического моделирования. Он, естественно, должен быть компетентен в предметной области конечного пользователя и результатом его работы обычно является разработка требований к программному обеспечению.

– **Архитектор программного** обеспечения решает задачи выбора структуры программного обеспечения, технологий, используемых для обработки, хранения и передачи данных, согласования действий отдельных исполнителей, дальнейшего развития и масштабирования системы и т. д.

– Основной задачей **прикладного программиста** является реализация принятых системных и архитектурных решения с использованием соответствующих технологий и языков программирования.

– **Системный программист** обеспечивает управление основными компонентами вычислительной системы и связь прикладного программного обеспечения с ресурсами используемой вычислительной техники.

Приведенный перечень далеко не окончателен и отражает только тот факт, что разработка программного обеспечения всегда носит коллективный характер. Данные в системе могут рассматриваться с различных позиций. Их интерпретация существенно зависит от уровня представления. Естественно, что каждому специалисту

важны определенные свойства данных и возможности их представления и обработки. Эти аспекты изучаются в отдельной дисциплине «структуры и алгоритмы обработки данных» [3, 4]. Однако на всех уровнях представления данных в компьютерных системах можно выделить общие особенности. Достаточно общими понятиями являются простейшие типы данных и составные структуры данных, хотя, в ряде случаев они имеют особенности использования на различных уровнях представления данных.

3.2. Простейшие (примитивные) типы данных

Простейшие типы данных составляют основу для построения других, более сложных структур данных. Из простейших данных могут быть построено множество сложных структур, используемых на различных уровнях представления данных в системе. Учитывая, что данные, как показано выше, по сути являются сообщениями, составленными из символов заданного алфавита, следует определить принципиальное отличие различных типов данных друг от друга.

Различные типы данных отличаются:

1. Операциями, которые можно выполнять над этими данными.
2. Возможными значениями или диапазонами значений.
3. Структурой хранения – выделение памяти, представление данных в ней и методом доступа.

К простейшим (примитивным) типам данных относятся:

– числовые данные;
– символьные данные;
– логические данные;
– указатели.

Числовые данные это множество всех чисел, используемых в системе.

Для числовых данных в общем случае определены арифметические операции сложения, вычитания, умножения и деления, а также операция арифметического сравнения двух чисел на равенство или отношение типа «больше», «меньше». Подчеркнем, что сравнение двух чисел выполняют следующим образом: из одного числа вычитается другое и анализируют результат. Если результат равен «0», то числа равны, иначе отношение определяется по знаку результата. Результат сравнения всегда является логическим данным.

К **символьным данным** относятся все символы алфавита, используемого в системе. В отличие от числовых данных для символов

не определены общепринятые операции. Считается [4], что базовой операцией, заданной для символов для символов, является их конкатенация или сцепление, в результате которой получается более сложная структура – строка символов

$$\text{"a"} \mid \text{"b"} = \text{"ab"}.$$

На практике большое значение имеет сравнение символов, которое часто используется при обработке текстовых массивов, например, при сортировке текстовых строк. Но арифметическое сравнение не применимо к символьным данным, так как для них не определены арифметические операции. Поэтому существует два подхода к сравнению символов:

– сравнение в лексикографическом порядке в соответствии с используемым национальным алфавитом, когда все символы алфавита пронумерованы и сравниваются порядковые номера символов в алфавите;

– сравнение по коду внутреннего представления символа в памяти вычислительной машины, подробно рассмотренного в разделе 10. В этом случае в сравнении участвуют двоичные числа, приписанные каждому символу используемой кодовой таблицей.

Результаты сравнения этими способами могут не совпадать.

Логические данные могут принимать только 2 значения «ИСТИНА» («True») или «ЛОЖЬ» («False»), часто обозначаемые как «логическая 1» и «логический 0».

К основным операциям, применяемым к логическим данным, относятся известные логические функции - конъюнкция, дизъюнкция, инверсия, сумма по модулю 2 и другие. Операция сравнения для логических данных имеет только применительно к сравнению на равенство.

Представление логических данных и операции над ними в вычислительной машине имеют свои особенности, связанные с тем, что хранимые данные в памяти являются многоразрядными двоичными числами, а для логического данного формально необходим только один разряд. Поэтому часто используются поразрядные логические операции, применимые в машинному слову. Эти особенности более подробно рассмотрены в разделе 11.

Указатели это данные, значением которых является адрес размещения объекта в вычислительной системе.

Обычно термин указатель применяют в языках программирования для обозначения адреса размещения данного в памяти вычислительной машины. Тогда значение указателя – это целое число без

знака, к которому применимы только операции сложения, вычитания и сравнения.

В общем случае к указателям относятся любые данные, указывающие местоположение объекта. Это могут быть ярлыки, имена файлов, URL и IP адреса и другие. Однако, особенности их использования определяются конкретной системой и выходят за рамки настоящего пособия.

3.3. Основные составные структуры данных – массивы и записи

На практике существенное значение имеют более сложные, составные структуры данных, использующие в качестве элементов простейшие типы данных. Изучение этих структур является предметом отдельной дисциплины [4], поэтому кратко остановимся на двух составных структурах данных, наиболее часто используемых в информационных технологиях.

Массив – это поименованная совокупность однотипных элементов, упорядоченных по индексам, определяющих положение элемента в массиве.

Количество используемых индексов определяет размерность массива. Чаще всего используют одномерные, двумерные и трехмерные массивы, хотя количество измерений в массиве принципиально не ограничено. Обычно массив занимает непрерывную область памяти. Обращение к элементу массива выполняется по индексу.

Для элементов массива применимы все операции, определенные их типом. Однако ко всему массиву могут быть применены и специфические операции. Например, присваивание одному массиву значения другого, сравнение массивов. Одномерный массив может рассматриваться, как вектор и тогда к нему применимы операции векторной алгебры. Двумерный массив может быть интерпретирован как матрица и к нему применимы операции специфические для матриц.

Составной структуре данных «**запись**» во многих языках программирования соответствует тип данных «структура».

Запись представляет собой составную структуру данных, состоящую из упорядоченной последовательности полей, каждое из которых содержит данное определенного типа. Запись, как правило, занимает непрерывную область памяти. Размер памяти, занимаемой записью, равен сумме размеров полей, составляющих запись.

4. ЧИСЛОВЫЕ ДАННЫЕ

4.1. Основные виды чисел

Одним из основных понятий математики является ***число***, используемое для количественной характеристики объектов, их нумерации, сравнения объектов и их частей. Изучение видов и свойств чисел проводится в отдельном разделе математики – «Теории чисел», в которой все возможные числа делятся на категории. В каждой категории вся совокупность возможных чисел образует множество.

Выделяют следующие основные виды числовых множеств.

– **Натуральные числа**, возникающие естественным образом при счете. Это целые числа больше нуля. Международный стандарт ISO 80000-2:2009 устанавливает два вида натуральных чисел, обозначаемых обычно N:

N – натуральные числа, включая ноль {0, 1, 2, 3, ...}, так называемый «расширенный натуральный ряд»,

N* – натуральное число без нуля {1, 2, 3, ...}.

Натуральные числа замкнуты относительно операций сложения и умножения, в которых результат тоже натуральное число. Сложение и умножение натуральных чисел коммутативны и ассоциативны. Операции вычитания и деления не замкнуты, так как определены не для всех пар натуральных чисел. Операция умножения дистрибутивна относительно сложения и вычитания.

В программных системах натуральному числу соответствует понятие целого числа без знака, причем, в отличие от математического понятия, ноль считается положительным числом.

Так, например, в языке C/C++ натуральное число может быть задано объявлением переменных:

```
unsigned char var_a;
unsigned int var_b;
```

Целые числа представляют собой множество всех отрицательных и положительных целых чисел на числовой оси и могут быть получены как разность двух натуральных чисел. Целые числа замкнуты относительно сложения, вычитания и умножения, но не деления. При их записи принято указывать знак целого числа символами «-» и «+», знак «+» часто опускают.

Для натуральных и целых чисел вводят отдельную операцию деления – деление с остатком.

В программировании этим числам соответствует понятие «целое со знаком». Например, такие данные могут быть описаны в языке C/C++:

```
signed char var_a;
signed int var_b;
```

– **Рациональные числа** – это числа, которые можно представить в виде отношения целого и натурального чисел, они замкнуты относительно всех четырех арифметических операций – сложения, вычитания, умножения и деления, кроме деления на ноль.

– **Действительные (вещественные) числа** представляют собой расширение множества рациональных чисел числами, которые не могут быть получены как отношение целых чисел.

Рациональные и действительные числа имеют целую и дробную части, которые при записи принято разделять знаками «точка» или «запятая». В этой группе особо следует выделить правильные дроби, в которых целая часть равна нулю. Они имеют важное значение в вычислительной технике при записи чисел с ***фиксированной точкой (запятой)***. При записи правильной дроби перед точкой, разделяющей целую и дробную части, записывают незначащий «ноль», однако следует иметь в виду, что этот «ноль» используется для удобства восприятия числа и не является символом, хранимым в памяти вычислительной машины.

В современном программировании действительные числа чаще представляются в форме с ***плавающей точкой***, например форматы `float`, `double`, `long double` в языке C/C++.

– **Комплексные числа** являются дальнейшим расширением действительных чисел и широко используются в математике, инженерной практике и других областях.

Кроме того, в теории чисел выделяют специальные виды чисел, рассмотрение которых выходит за рамки настоящего пособия.

4.2. Позиционная система счисления

Запись чисел всегда выполняется в виде некоторой последовательности символов, определяемой ***системой счисления***.

Система счисления – способ записи чисел с помощью ограниченного набора символов и совокупность правил (алгоритмов) сопоставления этим записям реальных значений чисел.

Символы, используемые для записи числа, часто называют цифрами данной системы счисления. Однако, для записи числа могут использоваться не только символы цифр, но и, например, символ

знака числа, точки или запятой, указывающих разделение числа на целую и дробную части.

Системы счисления могут быть позиционными и непозиционными, иногда используют смешанные системы счисления.

Для записи чисел в конкретной системе счисления используется алфавит некоторого размера. Этот алфавит не обязательно должен состоять из символов цифр, имеет значение только размер алфавита. Например, в китайском языке, для обозначения цифр используются соответствующие иероглифы.

Непозиционной системой называется такая система, в которой значение символа не зависит от его места расположения в числе. Для образования числа в непозиционной системе счисления используются операции арифметического сложения и вычитания. Примером непозиционной системы счисления является римская:

$$XVI = X + V + I = 10 + 5 + 1 = 16_{10},$$
$$IX = X - I = 10 - 1 = 9_{10}.$$

Примечание: *Здесь и далее будем при необходимости для цифр в качестве индекса указывать основание системы счисления.*

В **позиционной** системе счисления значение каждого символа алфавита зависит от его места расположения в числе. Справедливо следующее представление действительного положительного числа в позиционной системе счисления:

$$\mathbf{x(q) = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + \ldots}$$
$$\mathbf{+a_1q^1 + a_0q^0 + a_{-1}q^{-1} + \ldots + a_{-m}q^{-m} + \ldots,} \qquad (3)$$

где x(q) – число в заданной системе счисления; q – основание системы счисления, в позиционной системе счисления равен размеру алфавита, используемого для записи числа; q^i – вес i-того разряда числа; a_i – разрядный множитель – один из символов алфавита, используемого для записи числа, которому приписано соответствующее числовое значение; n – количество целых разрядов числа; m – количество дробных разрядов числа.

В общем случае количество разрядов дробной части числа не ограничено и равно бесконечности. Однако, в зависимости от решаемых задач, на практике всегда задают количество значимых дробных разрядов в представлении числа, называемого ***точностью*** представления.

Отметим некоторые особенности позиционной системы счисления:

1) основание системы счисления всегда натуральное число, большее 1 и равно 10 в этой системе счисления (например, $q = 2_{10} = 10_2$, $q = 8_{10} = 10_8$, $q = 16_{10} = 10_2$ и т. д.);

2) для выполнения основных арифметических операций – сложения, вычитания, умножения, деления и деления с остатком, достаточно задать только таблицы сложения и умножения;

3) в алфавите, используемом для записи числа, отсутствуют символы, отличные от символов цифр. Например, в него не входят символы «+», «–», «.», «,».

4) позиционной системе счисления соответствует запись числа в форме (3), называемой развернутой записью. В этой записи нет необходимости указывать разделитель целой и дробной частей, все определяется весом разрядов. На практике мы пользуемся сокращенной записью числа в виде последовательности разрядных множителей. Это естественно при использовании десятичной системы счисления, когда вес каждого разряда мы знаем, начиная с уроков арифметики в школе – единицы, десятки, сотни и т. д. При использовании другой системы счисления такая запись может вызвать затруднения;

5) сдвиг числа на один разряд влево или вправо соответствует умножению или делению числа на основание системы счисления, соответственно.

Многочлен (3) может быть записан в виде вычислительной схемы Горнера [5], используемой для построения алгоритмов перевода чисел из недесятичных систем счисления в десятичную. Для целого числа

$$\mathbf{x(q) = (...((a_{n-1}q + a_{n-2})q + a_{n-3})q + ... + a_1)q + a_0.}$$

Очевидно, что используемая система счисления определяет набор правил (алгоритмов) выполнения операций над числами. Поэтому, важное значение имеет правильное представление чисел и преобразование чисел в различных системах счисления.

Наибольшее распространение, кроме десятичной, в вычислительной технике имеют системы счисления с основаниями 2, 8, 16 – двоичная, восьмеричная, шестнадцатеричная. Для записи чисел в этих системах по возможности используют символы цифр.

1) Двоичная система счисления:

$$q = 2,\ A = \{0,1\}.$$

2) Восьмеричная система счисления:

$$q = 8, A = \{0, 1, 2, 3, 4, 5, 6, 7\}.$$

3) Для записи числа в шестнадцатеричной системе счисления используются десять символов цифр и символы латинского алфавита A, B, C, D, E, F, которыми приписывают числовые значения 10, 11, 12, 13, 14, 15, соответственно:

$$q = 16, A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}.$$

При работе с шестнадцатеричными числами в выражении (3) следует использовать десятичные значения символов A, B, C, D, E, F – (10), (11), (12), (13), (14),(15), соответственно.

Принятая запись шестнадцатеричного числа приводит к тому, что число может начинаться с символа латинского алфавита, а это автоматически воспринимается многими компиляторами не как число, а как имя объекта данных. Поэтому разные системы программирования накладывают дополнительные требования к форме записи шестнадцатеричных чисел.

В языке ASSEMBLER в конце записи числа указывается символ «h» (hexadecimal), а впереди записывается незначащий «0». Другие реализации языка требуют префикса «$» перед числом.

В языке C/C++ принята форма «0x*шестнадцатеричное_число*», например, 0xA9B3. Допускаются как прописные, так и строчные символы.

4.3. Перевод чисел из одной системы счисления в другую

Пусть необходимо перевести число из системы счисления $q_и$ в систему счисления $q_к$. В зависимости от того, какие значения имеют $q_и$ и $q_к$, в инженерной практике используются различные правила:

1) перевод числа из десятичной системы в систему с любым основанием – $q_и = 10$, $q_к \neq 10$:

1.1) перевод целого числа,

1.2) перевод правильной дроби,

1.3) перевод числа, имеющего целую и дробную часть;

2) перевод числа из системы с любым основанием ($q_и \neq 10$) в десятичную ($q_к = 10$);

3) перевод числа произвольной недесятичной системы с основанием $q_и \neq 10$ в систему с основанием $q_к \neq 10$;

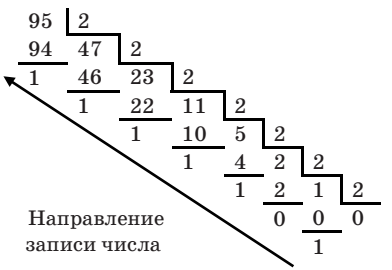
4) перевод числа из двоичной системы счисления $q_и = 2$, в систему счисления $q_к = 2^p$, где $p = \{2, 3, 4, ...\}$;

5) перевод числа из системы счисления $q_и = 2^p$, где p = {2, 3, 4, ...}, в двоичную систему счисления $q_к = 2$.

Правило 1.1. *Для перевода целого числа из десятичной системы счисления ($q_и = 10$) в недесятичную систему с основанием $q_к \neq 10$, число нужно последовательно делить на основание $q_к$ до тех пор, пока не будет получена целая часть частного, равная 0, то есть будет получен остаток от деления, меньший $q_к$. Число в системе счисления с основанием $q_к$ записывается в виде упорядоченной последовательности остатков от деления в порядке, обратном получению остатков, то есть старшей цифрой числа будет последний остаток.*

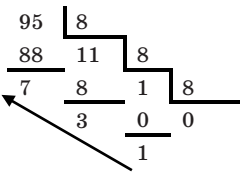
Пример: перевести число 95_{10} в следующие системы счисления:

а) двоичную:



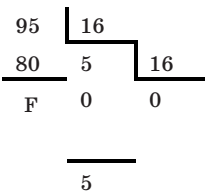
Ответ: $95_{10} = 1011111_2$

б) восьмеричную :



Ответ: $95_{10} = 137_8$

в) шестнадцатеричную:



Ответ: $95_{10} = 5F_{16}$

Правило 1.2. *Для перевода правильной дроби из десятичной системы счисления ($q_и = 10$) в систему с основанием $q_к \neq 10$ число нужно последовательно умножать на основание $q_к$ (причем умножению подвергаются только дробные части результатов промежуточных произведений) до тех пор, пока не будет обеспечена заданная точность представления числа. Дробь в системе счисления с основанием $q_к$ записывается в виде упорядоченной последовательности целых частей произведений в порядке их получения.*

– Если требуемая точность перевода в $q_к$ составляет m разрядов после запятой, отделяющей дробную часть, то число указанных

последовательных произведений (то есть цифр в представлении дроби) равно m+1. Затем результат округляется до m разрядов.

– Если на некотором шаге получения промежуточного произведений дробная часть числа становится равной 0, то процесс преобразования на этом заканчивается, так как все остальные цифры в представление дроби будут равны 0, а результат дополняется до m разрядов незначащими нулями.

Чаще всего используется округление к ближайшему по модулю числу. Для двоичной системы счисления, если значение ***m + 1*** разряда равно «0», то оно отбрасывается, если «1», предыдущим разрядам прибавляется «1».

Особенностью перевода правильной дроби является то, что не все дроби могут быть представлены в позиционной системе счисления точно. Точное представление возможно только в случаях, когда знаменатель дроби является степенью основания системы счисления. В остальных случаях требуется бесконечное число разрядов в дробной части числа, что, естественно, приводит к ошибке округления. Поэтому при переводе всегда задается точность представления результата ***m***, определяющая количество значащих цифр после точки.

Пример: перевести число $.95_{10}$ в следующие системы счисления, с точностью представления m . Для $q_к$ =2, m = 6; $q_к$ = 8, m = 2; $q_к$ = 16, m = 2, соответственно:

а) двоичную:

0	95
	2
1	90
	2
1	80
	2
1	60
	2
1	20
	2
0	40
	2
0	80
	2
1	60

б) восьмеричную:

0	95
	8
7	60
	8
4	80
	8
6	40

в) шестнадцатеричную:

0	95
	16
F	20
	16
3	20
	16
3	20

Ответ: $.95_{10} = .1111001_2 = .111101_2$;
$.95_{10} = .746_8 = .75$;
$.95_{10} = .F33_{16} = .F3_{16}$.

Обратите внимание, что в приведенной записи точка является разделителем целой и дробной части, а перед точкой отсутствует «0», так как он не является частью числа. Обычно мы записываем правильную дробь, указывая впереди «0», означающий только, что целая часть числа отсутствует.

Правило 1.3. *При переводе неправильной дроби из десятичной системы счисления в систему с основанием $q_к \neq 10$ отдельно переводится целая и дробная части числа.*

Пример: перевести число 123.58_{10} в восьмеричную систему счисления

$$123.58_{10} = 123_{10} + 0.58_{10} = 173_8 + 0.45_8 = 173.45_8$$

Правило 2. *Для перевода числа из недесятичной системы счисления с основанием $q_u \neq 10$ в десятичную, $q_к = 10$, пользуются формулой разложения в ряд (1). Выражение (1) справедливо как для целых, так и для дробных чисел.*

Пример: представить числа: а) 111011.011, б) 154.31, в) A2B.3C в десятичной системе счисления.

$$111011.011_2 = 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 0*2^{-1} + \\ + 1*2^{-2} + 1*2^{-3} = 32 + 16 + 8 + 2 + 1 + 1/4 + 1/8 = 59.375_{10};$$

$$154.31_8 = 1*8^2 + 5*8^1 + 4*8^0 + 3*8^{-1} + 1*8^{-2} = \\ = 64 + 40 + 4 + 3/8 + 1/64 = 108.390625_{10};$$

$$A2B.3C_{16} = A*16^2 + 2*16^1 + B*16^0 + 3*16^{-1} + C*16^{-2} = \\ = 3840 + 32 + 11 + 3/16 + 12/256 = 3883.234375_{10}.$$

Для двоичной системы счисления для перевода в десятичную можно пользоваться следующим правилом:

Сложить веса тех разрядов, для которых в записи двоичного числа «1».

Веса разрядов легко запомнить – 1, 2, 4, 8, 16, 32.....

Правило 3. *При переводе чисел из системы счисления с основанием $q_u \neq 10$ в систему с основанием $q_к \neq 10$ выполняется промежуточное преобразование в десятичную систему.*

Правило 4. *Для перевода двоичного числа $q_u = 2$ в число с основанием системы счисления, равным целой степени 2, $q_к = 2^p$, где $p = \{2, 3, 4, ...\}$ необходимо разбить исходное двоичное число на группы*

из p разрядов. Разбиение выполняется для целой части числа справа на лево от запятой, а для дробной – слева на право от запятой. При необходимости группы дополняются незначащими нулями. Число в новой системе счисления записывается как последовательность символов, соответствующих каждой группе.

Пример: $11000111001010111101011011 0.0101110101_2$ перевести в шестнадцатеричную систему счисления.

0110	0011	1001	0101	1110	1011	0110	.0101	1101	0100
6	3	9	5	E	B	6	5	D	4

Ответ: $11000111001010111101011011 0.0101110101_2 = 6395EB6.5D4_{16}$.

Для записи шестнадцатеричного числа в группе достаточно сложить веса тех разрядов двоичного числа, которые имеют единичное значение. Значения веса разряда в группе 8, 4, 2, 1 легко запомнить.

Из приведенного примера следует, что восьмеричная и шестнадцатеричная системы счисления используются только для того, чтобы компактно отображать значения громоздких двоичных чисел, хранящихся в памяти вычислительной машины.

Правило 5. *Перевод из числа $q_u = 2^p$, где $p = \{2, 3, 4, ...\}$ в двоичное число $q_к = 2$ выполняется последовательной записью двоичных чисел, соответствующих каждому символу исходного числа.*

Пример: $A2E.C1D_{16} = 101000101110.110000011101_2$.

5. ФОРМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В РАЗРЯДНОЙ СЕТКЕ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ

5.1. Разрядная сетка. Представление целых чисел без знака

Любое данное может размещено в целом числе ячеек памяти. Например, в большинстве современных цифровых устройств размер ячейки памяти составляет 1 байт (8 двоичных разрядов) и, в зависимости от формата данного, для него отводится целое, обычно четное, число байт. Количество двоичных разрядов, выделенных для хранения данного, называют разрядной сеткой. Разрядная сетка для определенного типа данных имеет фиксированный размер, состоящий из **n** разрядов. Размер разрядной сетки в программах задается на этапе объявления переменных. Имя переменной (идентификатор) при выполнении программы связывается с адресом ее размещения в памяти, а тип определяет алгоритмы вычислений, применимые к этим данным. Алгоритмы, как правило, реализуются отдельными библиотечными функциями системы программирования.

Например, в языке C/C++ объявление переменной

```
unsigned char var_a;
```

означает, что для переменной выделяется память в 1 байт, обращение к этой ячейке будет выполняться по адресу, имеющему символическое имя `var_a`, переменная `var_a` является натуральным числом (числом без знака), а множество операций над содержимым этой ячейки определится набором функций, описанных в директивах `#include`. При этом необязательно это будут только арифметические операции, программист может предусмотреть и другие, используя соответствующие библиотечные функции. Обязательным требованием любой системы программирования является одинаковость типов, а, следовательно, размеров данных, используемых в выражениях. Поэтому при вычислениях либо автоматически, либо явно осуществляется операция «приведение типов», в которой переменные меньшей разрядности преобразуются к формату с большей разрядностью.

В общем случае разрядная сетка представляется словом из **n** двоичных разрядов (рис. 5).

При двоичном представлении чисел в позициях разрядной сетки могут быть записаны только символы A = {0,1}. Таким образом, однозначно в пределах разрядной сетки можно представить только натуральное, целое без знака число. Именно на такое представление ориентированы команды процессора вычислительной машины.

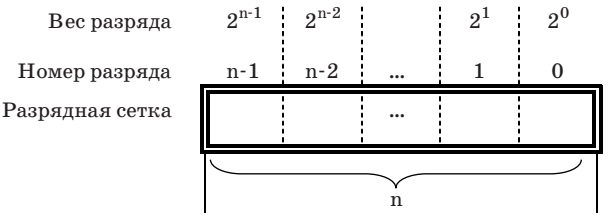


Рис. 5. Разрядная сетка

Алгоритмы обработки других видов чисел реализуются библиотечными функциями, включенными в систему программирования и, следовательно, могут быть различными в различных системах программирования в зависимости от класса решаемых задач.

Общим подходом к представлению более сложных чисел является разбиение разрядной сетки на поля, имеющие различное назначение, что учитывается в соответствующих алгоритмах вычислений.

5.2. Представление целых чисел со знаком

В представлении этих чисел необходимо указать знак числа. Для указания знака разрядная сетка разбивается на два поля – знак и значащая часть числа. В двоичной системе счисления знак кодируется для положительных чисел, включая ноль, символом «0», а для отрицательных символом «1». Следует иметь в виду, что для знаков не определены арифметические операции, поэтому существует ряд особенностей в алгоритмах обработки чисел со знаком. Под поле знака, как правило, выделяется старший разряд разрядной сетки, называемый «***знаковый разряд***» (рис. 6).

Символы в знаковом разряде обозначают знак, а не число, и, соответственно, в общем случае не могут участвовать в арифметических операциях.

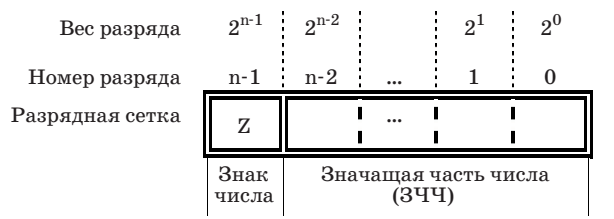


Рис. 6. Разрядная сетка для целого числа со знаком в двоичной системе счисления

При использовании недвоичной системы счисления в знаковом разряде можно записать более чем 2 значения, обозначающие «+» и «–». Тогда необходимы дополнительные соглашения о кодировании знака числа.

Так, например, в ряде случаев используется упакованный десятичный формат числа – BCD (Binary-Coded Decimal). В этом формате для хранения десятичного числа выделяется целое число байт, каждый байт разбивается на два четырехразрядных поля – тетрады, в которых хранится двоичное представление соответствующего десятичного разряда. В этом формате знак числа определяет старший бит старшей тетрады в записи числа.

5.3. Числа с фиксированной точкой

Для представления действительных чисел, имеющих дробную часть, в разрядной сетке необходимо указать разделитель целой и дробной части – символ «точка» или «запятая». Эти символы не входят в алфавит, используемый для записи числа, и поэтому не могут быть использованы. Отметим, что в развернутой форме записи числа (3) необходимости указания символа «точка» нет. В разрядной сетке запись числа осуществляется в сокращенной форме, где вес разряда определяется его положением – номером разряда. О положении разделителя целой и дробной части необходимы дополнительные соглашения, определяющие алгоритмы вычислений, из которых следует особо выделить два, используемых для внутреннего представления чисел в вычислительной машине.

1. Целое число можно рассматривать как действительное, у которого отсутствует дробная часть. Тогда точка, разделяющая целую и дробную части, подразумевается справа от самого младшего разряда числа. Дробная часть находится за пределами разрядной сетки.

2. Целая часть числа равна нулю и, следовательно, число является правильной дробью. Тогда точка, разделяющая целую и дробную части, подразумевается между знаковым разрядом и значащей частью числа. Такое представление называется **«число с фиксированной точкой».**

Число с фиксированной точкой представляется следующим образом:

$$X_{ф.т.} = X^{*}К_{м}, \qquad (4)$$

где $X_{ф.т.}$ – машинное представление числа с фиксированной точкой; X – исходное число, $К_{м}$ – масштабный коэффициент, который

выбирается из условий конкретной разрядной сетки и не должен допускать выхода исходных чисел и результатов вычислений за пределы допустимого диапазона. Обычно масштабный коэффициент кратен основанию системы счисления, что позволяет свести операции умножения и деления числа на масштабный коэффициент к более быстрым операциям сдвига числа.

Масштабный коэффициент должен быть единым для всех обрабатываемых в машине чисел и получаемых результатов, он хранится отдельно от представляемых чисел и учитывается при выдаче конечного результата.

Таким образом, число с фиксированной точкой всегда является правильной дробью.

Если при вычислениях результат выходит из значащей части разрядной сетки, т. е. результат имеет целую часть, отличную от нуля, фиксируется ситуация ***переполнения разрядной сетки***. В вычислительной машине при возникновении переполнения устанавливается бит **OF** в регистре состояния и, как правило, возникает прерывание выполнения программы. Алгоритмы обработки переполнения зависят от назначения и особенностей программного обеспечения и выбираются на стадии его разработки.

При вычислениях с числами с фиксированной точкой возможно получение результата с разрядностью дробной части числа, превышающей разрядность значащей части разрядной сетки. В этом случае младшие разряды результата отбрасываются, что приводит к появлению погрешности вычислений. Такая ситуация возникает при умножении двух чисел, когда разрядность результата принципиально может быть в два раза больше разрядности операндов. Зафиксировав точку перед знаковым разрядом, мы гарантированно исключаем возможность переполнения разрядной сетки при умножении. Младшие разряды результата, не входящие в разрядную сетку, просто отбрасываются.

5.4. Числа с плавающей точкой

Представление чисел в форме с плавающей точкой использует экспоненциальную форму записи числа, что позволяет избежать масштабирования исходных чисел и использовать более гибкие алгоритмы вычислений. Практически во всех современных процессорах для арифметических операций над числами с плавающей точкой используется специальное устройство – математический сопроцессор для чисел с плавающей точкой.

Число с плавающей точкой [6] представляется в виде

$$X = M*E^P, \quad (5)$$

где E = q – основание системы счисления, P – порядок числа, M – мантисса числа.

В литературе такая запись часто называется логарифмической.

Пример:

$X_{10} = 525*10^0 = 5.25*10^2 = 0.525*10^3$ – запись десятичного числа в рассматриваемом формате.

Из этого примера следует, что одно и тоже число в формате с плавающей точкой может иметь различную запись.

В представлении (5) порядок является целым числом со знаком. Мантисса может иметь как целую, так и дробную части, а ее знак определяет знак числа.

Как следует из выражения (5) запись действительного числа допускает различные варианты, поэтому для обеспечения совместимости различного программного обеспечения при работе на различных аппаратных платформах и при обмене данными между различными приложениями машинное представление чисел с плавающей точкой должно быть стандартизовано. Это регламентирует международный стандарт IEEE 754 (754-2008 – IEEE Standard for Floating-Point Arithmetic), подробное рассмотрение которого выходит за рамки настоящего пособия.

Стандарт IEEE 754 предусматривает три основных формата представления двоичных чисел:

– число одинарной точности (`float` в языке C/C++);

– число двойной точности (`double` в языке C/C++);

– расширенный формат (`long double` в языке C/C++) не является обязательным и используется не во всех системах программирования.

Рассмотрим только основные особенности машинного представления двоичных чисел с плавающей точкой, общие для различных форматов представления чисел.

Разрядная сетка размером **n** делится на три поля (рис.7) для указания значений знака числа, порядка разрядностью $\mathbf{n_p}$ и мантиссы разрядностью $\mathbf{n_m}$. Точка, разделяющая целую и дробную части мантиссы подразумевается между полями порядка и мантиссы. Так, например, для хранения числа в формате с одинарной точностью (`float`) отводится 4 байта, из которых старший бит определяет знак числа, 8 разрядов – порядок и 23 разряда мантиссу. Точка, разделяющая целую и дробную части мантиссы подразумевается.

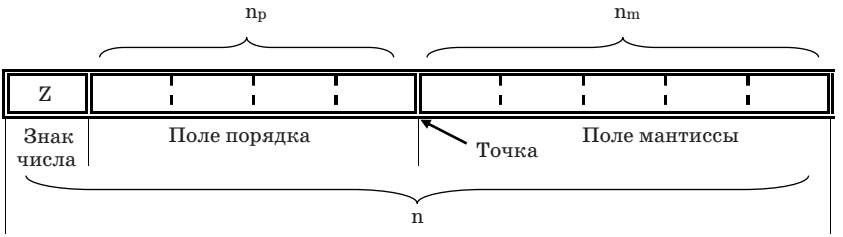


Рис. 7. Общее представление двоичного числа в разрядной сетке по стандарту IEEE 754

Особенности выполнения арифметических операций в вычислительной машине и стремление к сокращению времени их выполнения определяют формы заполнения полей разрядной сетки.

1. Мантисса числа должна быть нормализованной, т. е. отвечать условию

$$1 \leq M_н < E,$$

для двоичного числа

$$1 \leq M_н < 2_{10}.$$

Таким образом, нормализованная мантисса двоичного числа всегда имеет целую часть, равную 1. Поэтому целая часть нормализованной мантиссы не хранится в разрядной сетке, а ее единичное значение учитывается либо на аппаратном уровне в математическом сопроцессоре, либо в соответствующих алгоритмах вычислений, ее называют «скрытая единица». В результате появляется возможность увеличить диапазон представления вещественных чисел за счет увеличения разрядности мантиссы. Это справедливо для чисел с одинарной и двойной точностью. Расширенный формат, как внутренний формат представления любого типа числа в сопроцессоре, содержит целую единицу в явном виде.

В поле мантиссы разрядной сетки хранится модуль дробной части числа.

Особым случаем является нулевое значение числа. «Ноль» в таком представлении может иметь как знак «+», так и «−». Нулевую мантиссу нельзя нормализовать, поэтому вводится понятие «машинный ноль», за которое принимают число с нулевым значением мантиссы и порядка.

Процесс нормализации вещественного двоичного числа ничем не отличается от нормализации десятичного вещественного числа. Например, десятичное число 1234.567 в нормализованном виде

выглядит так: $1.234567*10^3$. Как видно, в процессе нормализации десятичная точка переносится влево или вправо так, чтобы перед ней находилась только одна десятичная цифра. При этом значение показателя степени определяет количество цифр, на которое нужно передвинуть десятичную точку вправо (если показатель положителен) или влево (если показатель отрицателен), чтобы получить истинное значение числа.

Аналогично для двоичного числа точку, разделяющую целую и дробную части необходимо сдвигать влево или право до тех пор, пока не будет получена целая часть числа, равная 1. На каждом шаге сдвига порядок числа увеличивается (при сдвиге точки влево) или уменьшается (при сдвиге точки вправо) на единицу. Например, если при нормализации точка была перенесена на 4 разряда влево, порядок числа должен быть увеличен на 4.

Примеры нормализации мантиссы двоичного числа:

а) $111011.101 = 1.11011101*E^{0101}$;

б) $1.110111 = 1.110111*E^{0}$;

в) $0.0011011 = 1.1011000*E^{1011}$.

В приведенных примерах указан двоичный порядок как целое число со знаком. E – экспонента, равная в данном случае 2_{10} или 10_2. Основание системы счисления подразумевается и нигде не хранится, поэтому записи числа $1.11011101*E^{010}$ и $1.11011101*10^{010}$ эквивалентны.

Стандарт IEEE 754 требует, чтобы в поле порядка всегда указывалось только целое число без знака. Это позволяет упростить алгоритмы и, соответственно, уменьшить время вычислений. Сложение и сравнение целых чисел без знака выполняется быстрее.

Поэтому в поле порядка хранится смещенный на $(2_p^{n-1} - 1)$ порядок $P_{см}$ (рис. 8), называемый характеристикой

$$P_{см} = P + (2_p^{n-1} - 1).$$

Характеристика всегда целое число без знака.

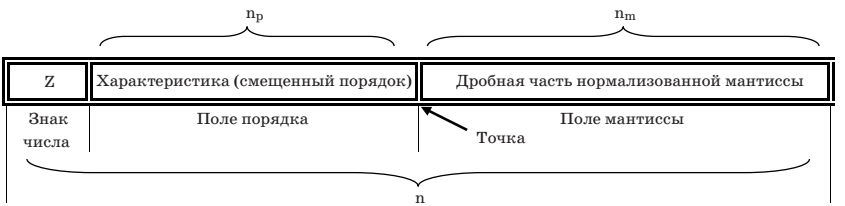


Рис. 8. Машинное представление двоичного числа в разрядной сетке по стандарту IEEE 754, целая часть мантиссы равна 1 и подразумевается

Смещение порядка учитывается при выводе окончательного результата вычислений.

Для приведенных выше примеров машинное представление при $\mathbf{n_p} = 4$ и $\mathbf{n_m} = 8$:

а) $111011.101 = .11011101*E^{1100}$;

б) $1.110111 = .11011100*E^{0111}$;

в) $0.0011011 = .10110000*E^{0100}$.

Обобщенный алгоритм преобразования вещественного десятичного числа в двоичное число с плавающей точкой формата IEEE.

Этот алгоритм описывает последовательность действий, которые нужно предпринять, для преобразования вещественного числа в двоичное число формата IEEE.

1. Представить целую часть вещественного числа в двоичном виде и поставить после нее десятичную точку.

2. Преобразовать дробную часть вещественного числа в двоичную систему счисления с точностью $\mathbf{n_m}$ после точки.

3. Записать полученное значение в двоичной форме.

4. Нормализовать полученное двоичное число, определив значение показателя степени в порядке P.

5. Найти характеристику $P_{см} = P + (2^{n-1}_{p} - 1)$.

6. Записать значение характеристики в соответствующие биты формата.

7. Записать дробную часть нормализованной мантиссы в поле мантиссы, отбросив разряды, выходящие за пределы этого поля.

Примечание. Правила округления мантиссы могут отличаться в различных системах программирования.

8. Если число положительное, то в самый старший разряд представления следует записать 0, если отрицательное – то 1.

При записи вещественного числа в ограниченном формате разрядной сетки неизбежны погрешности.

Так, например, перевод десятичного числа 0.2_{10} в представлении `float` (C/C++) дает значение 0.200000003_{10}. Несмотря на то, что погрешность незначительна, это может привести к серьезным ошибкам при сравнении чисел.

Например, приведенные ниже фрагменты программ для систем MATLAB и C/C++ , дадут разные результаты.

В программе на зыке C/C++ переменная `var_b` получит значение 0, а в MATLAB – 1.

C/C++:

```
float var_a;
char var_b =0;
```

```
var_a = 0.2;
if ((var_a - 0.2) == 0) var_b = 1;
```

MATLAB:

```
var_a = 0.2;
var_b = 0;
if (var_a - 0.2) == 0
var_b = 1;
end
```

Ошибка в программе на C/C++ заключается в том, что в сравнении участвуют значения разных форматов `var_a - float`, константа 0.2 по умолчанию имеет формат `double`. При вычислении будет выполнено автоматическое приведение типов данных `float var_a = -0.200000003` будет приведено к типу `double` добавлением незначащих нулей. Соответственно, `(0.20000000300000000 - 0.20000000000000001) ≠ 0`.

6. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ СО ЗНАКОМ В ВЫЧИСЛИТЕЛЬНОЙ МАШИНЕ

6.1. Числа в прямом и дополнительном кодах

Для хранения чисел со знаком, как показано на рис. 6, в разрядной сетке выделяется отдельный разряд для хранения знака. Всегда необходимо учитывать, что в арифметических операциях могут участвовать только числа одинакового типа и, следовательно, должны быть представлены в **одинаковой** разрядной сетке. Реализация принятых в классической арифметике алгоритмов выполнения арифметических операций в вычислительной машине не эффективна.

Так, например, известное из классической арифметики правило сложения чисел со знаком:

«Для сложения чисел со знаком необходимо:

– сравнить знаки операндов;

– если знаки одинаковы, сложить операнды и присвоить результату их знак;

– если знаки разные, из большего по модулю числа вычесть меньшее по модулю и результату присвоить знак большего по модулю.»

Таким образом, операция сложения требует, кроме сложения и вычитания, последовательного выполнения ряда операций определения модуля числа, сравнения и присвоения знака, выполняемых до непосредственного сложения. С точки зрения времени выполнения этих операций алгоритм не эффективен и приводит к существенному его увеличению.

На практике, числа со знаком представляются либо в прямом, либо в дополнительном кодах. Формальная запись этих представлений приведена в табл. 1.

Приведенные в табл. 1 выражения предполагают, что число записано в разрядной сетке размером **n** в конкретной позиционной системе счисления с основанием **q**. Соответственно, число X занимает n – 1 разряд. Старший разряд разрядной сетки отводится под знак. Если q = 2, знак кодируется «0» для положительных чисел и «1» для отрицательных. Для q > 2 положительному числу соответствует знаковый разряд, равный 0, а отрицательному – наибольшее значение значащей цифры в данной системе счисления.

Например, для десятичного числа:

X = 57 – число без знака;

$+X_{пр} = 0_z57$ – число со знаком, его разрядность на 1 больше, «0» – знак «+»;

$-X_{пр} = 9_z57$ – «9» кодирует знак «–».

Таблица 1

Представление чисел для системы счисления с произвольным основанием q

Прямой код	Дополнительный код
Целое число: $X_{пр} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ (q-1) * q^{n-1} + \lvert X\rvert, \text{ при } X<0 \end{cases}$	Целое число: $X_{д} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ q^{n} - \lvert X\rvert, \text{ при } X < 0 \end{cases}$
Правильная дробь: $X_{пр} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ (q-1) + \lvert X\rvert, \text{ при } X < 0 \end{cases}$	Правильная дробь: $X_{д} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ q - \lvert X\rvert, \text{ при } X < 0 \end{cases}$

Такие соглашения позволяют интерпретировать знак, как число, разместить его в разрядной сетке и учитывать в алгоритмах вычислений. В частности, положительное число записывается как его модуль с разрядностью **n – 1** с нулевым значением знакового разряда, а отрицательное целое число в прямом коде можно представить как сумму модуля числа X и знаком, равным $\mathbf{(q-1)*q^{n-1}}$.

В приведенном примере **n** =3, $\mathbf{(q-1)*q^{n-1}} = 9*10^2$, а – $X_{пр} = 9_z00 + 0_z57 = 9_z57$.

Аналогично для правильной дроби **(q – 1)** в десятичной системе счисления, в которой вес старшего разряда, отводимого под знак, равен 10.

Представление числа в прямом коде соответствует привычной для практики записи числа со знаком. Поэтому, как правило, результаты вычислений представляются в прямом коде.

Достоинства прямого кода:

– простое получение кода отрицательных чисел;

– код значащей части числа является его модулем и совпадает с соответствующим числом без знака, что удобно для сравнения чисел;

– при заданной разрядной сетке количество положительных чисел равно количеству отрицательных.

Недостатки прямого кода:

– выполнение арифметических операций требует раздельной обработки значащей части числа и знака, для которого не определены арифметические операции. Это требует усложнения алгоритмов вычислений либо изменений в архитектуре процессора;

– число «0» имеет два значения – «+0» и «–0».

В рассмотренных выше числах с плавающей точкой мантисса представлена в прямом коде.

В современных вычислительных системах целые числа со знаком и числа с фиксированной точкой представляются в дополнительном коде.

Для дополнительного кода используется известное в математике понятие «арифметическое дополнение». **АРИФМЕТИЧЕСКОЕ ДОПОЛНЕНИЕ** числа A ($0 \le A < 1$) есть разность между единицей и этим числом. Арифметическое дополнение часто используется при логарифмических вычислениях. Иногда арифметическое дополнение рассматривают не до единицы, а до десяти – основания системы счисления.

В вычислительной технике арифметическое дополнение рассматривают как дополнение числа до значения, на единицу большего, чем максимальное число, которое может быть записано в разрядной сетке. Отрицательное число записывают как дополнение соответствующего положительного числа. В выражениях для отрицательных чисел в дополнительном коде $\mathbf{q^n}$ для целых чисел и **q** для правильной дроби по сути есть число, имеющее «1» в разряде, выходящем за пределы разрядной сетки и нулевые значения в остальных разрядах, т. е. на «1» большее, чем наибольшее число, которое может быть записано в разрядной сетке размером **n**.

Поясним преимущества использования дополнительного кода для целых чисел, представленных в **n** разрядной сетке.

$$A - B = A + (-B) = A + (q^n - B) - q^n = A + B_д - q^n .$$

q^n выходит за пределы разрядной и ее можно отбросить. Тогда

$$A - B = A + B_д.$$

Таким образом, операция вычитания сведена к сложению в дополнительном коде, для ее выполнения не требуется предварительного выделения модулей операндов и сравнений.

В двоичной системе счисления нет необходимости в вычислениях дополнения по приведенным в табл. 1 формулам.

Чтобы найти дополнение для числа в двоичной системе счисления необходимо выполнить инверсию всех разрядов и прибавить «1» к младшему разряду.

Во внутренней системе команд любого процессора есть команда «сложение с инверсией + 1», которая по сути реализует это правило. Таким образом, на сложение (вычитание) двух машинных слов расходуется один такт машинного времени.

Отметим особенности представления чисел в дополнительном коде:

– представление положительных чисел в прямом и дополнительном кодах совпадают;

– знаковый разряд может рассматриваться как число и, соответственно, его можно использовать в вычислениях вместе с разрядами значащей части числа;

– наиболее простой и быстрый алгоритм выполнения сложения чисел со знаком;

– чтобы изменить знак числа, записанного в дополнительном коде, достаточно взять дополнение;

– наличие только одного нуля;

– представление не симметрично – число отрицательных чисел на «1» больше, чем положительных.

Рассмотренные выше правила записи чисел в прямом и дополнительном кодах имеют общий характер и применимы к различным системам счисления. В современной практике в цифровых устройствах используется двоичная система счисления для которой формулы, приведенные в табл.1, могут быть существенно упрощены.

Примечание. В большинстве учебников и публикаций рассматривается кодирование чисел со знаком только для двоичной арифметики. Однако более общее представление полезно, так как известны разработки вычислительных машин, работающих в недвоичных системах счисления, например, троичной или десятичной. Десятичное представление чисел в формате «упакованного десятичного числа» (BCD) предусмотрено во многих современных процессорах и математических сопроцессорах, реализующих требования стандарта IEEE754 В табл. 2 приведена формальная запись двоичных чисел в прямом и дополнительном кодах.

Таблица 2

Представление чисел для двоичной системы счисления

Прямой код	Дополнительный код
Целое число: $X_{пр} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ (2^{n-1})_2 + \lvert X\rvert, \text{ при } X < 0 \end{cases}$	Целое число: $X_{д} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ (2^{n})_2 - \lvert X\rvert, \text{ при } X < 0 \end{cases}$
Правильная дробь: $X_{пр} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ 1 + \lvert X\rvert, \text{ при } X < 0 \end{cases}$	Правильная дробь: $X_{д} = \begin{cases} \lvert X\rvert, \text{ при } X \geq 0; \\ 2_2 - \lvert X\rvert, \text{ при } X < 0 \end{cases}$

Примеры

1. Представить в прямом и дополнительном десятичных кодах следующие числа $X = 78_{10}$ и $Y = 8_{10}$ и вычислить (X-Y) и (Y-X), ответ записать в прямом коде:

Значащая часть числа X без знака 2 разряда для записи числа со знаком требуется 3 разряда.

$$+X_{пр} = +X_{д} = 0_z78; \quad -X_{пр} = 900 + |X| = 9z78;$$

$$-X_{д} = 1000 - 078 = 9_z22;$$

при записи числа Y со знаком, добавляем не значащий ноль, чтобы разрядности операндов были одинаковы

$$+Y_{пр} = +Y_{д} = 0_z08; \quad -Y_{пр} = 900 + |Y| = 9_z08;$$

$$-Y_{д} = 1000 - 008 = 9_z92.$$

$$(X\text{-}Y)_{д} = 0_z78 + 9_z92 = 10_z70 = 0_z70.$$

Ответ: $(X\text{-}Y)_{пр} = 0_z70$.

В результате старшая «1» отбрасывается, так как не умещается в разрядную сетку.

$$(Y\text{-}X)_{д} = 0_z08 + 9_z22 = 9_z30.$$

Получаем отрицательное число в дополнительном коде. Переводим результат в прямой код. Для этого находим модуль числа и прибавляем к нему $(q - 1)*q^{n-1}$

$$(Y\text{-}X)_{пр} = (q - 1)*q^{n-1} + |X| = 9000 + (1000 - 9_z30) =.$$

Ответ: $(Y\text{-}X)_{пр} = 9_z70$.

2. Представить в прямом и дополнительном двоичном коде следующее числа десятичные и вычислить (X-Y) и (Y-X), ответ записать в прямом двоичном и десятичном кодах:

$X = 78_{10} = 1001110_2$, $Y = 8_{10} = 0001000_2$. (разрядности должны быть одинаковы!)

$$+X_{пр} = +X_{д} = 0_z1001110_2; \quad -X_{пр} = 1_z1001110_2, \quad -X_{д} = 1_z0110010_2;$$

$$+Y_{пр} = +Y_{д} = 0_z0001000_2; \quad -Y_{пр} = = 1_z0001000_2; \quad -Y_{д} = 1_z1111000_2;$$

$(X\text{-}Y)_{д} = 0_z1001110_2 = 0_z1000110_2 = 0_z70_{10}$ (старшая «1» отбрасывается)

$1_z1111000_2$

$10_z1000110_2$

Ответ: $(X\text{-}Y)_{пр} = 0_z1000110_2 = 0_z70$.

$(Y\text{-}X)_д = 0_z0001000 = 1_z0111010_2$

$1_z0110010_2$

$1_z0111010_2$

Ответ: $(Y\text{-}X)_{пр} = 1_z1000110_2 = 9_z 70$.

6.2. Обработка переполнения разрядной сетки при сложении двоичных чисел

Одна из особенностей выполнения арифметических операций в вычислительной машине состоит в возможности переполнения разрядной сетки фиксированного размера. Размер разрядной сетки определяется только аппаратными особенностями процессора и не может быть произвольным.

При выполнении арифметических операций размер результата может превышать размер операндов. Так, например, при сложении двух чисел разрядностью **n** может быть получен результат размером **n+1**, а при умножении – размером **2n**. В этих случаях возникает **переполнение разрядной сетки**.

Для фиксации признаков результата вычислений в процессоре предусмотрен специальный регистр, называемый флаговый регистр (*flags*) или регистр состояния. Отдельные разряды этого регистра фиксируют признаки результата вычислений. Эти признаки используются для управления вычислительным процессом и при выполнении команд условного перехода. Для обработки переполнений определяющее значение имеют разряды регистра *flags* CF (Carry Flag) и OF (Overflow Flag).

В разряд CF записывается «1» при возникновении переноса из знакового разряда разрядной сетки и «0» при его отсутствии.

Разряд OF устанавливается в «1» при переполнении разрядной сетки.

Суть механизма возникновения переполнения разрядной сетки состоит в том, что значащая часть результата, имеющего размер на «1» больше, чем разрядность операндов, занимает знаковый разряд, а знак «вытесняется» за пределы разрядной сетки, занимая бит CF регистра флага. Таким образом, при возникновении переполнения обеспечивается возможность получения значения результата, но дальнейшее его участие в операциях в рамках заданной разрядной сетки не возможно. Реакция программы на переполнение задается на этапе ее проектирования и определяется особенностями обработки данных в конкретной предметной области. В типовом случае происходит аварийное завершение программы.

Рассмотрим механизм возникновения переполнения разрядной сетки при сложении чисел.

При поразрядном сложении многоразрядных чисел одинаковой разрядности возможно возникновение переноса из текущего разряда в соседний старший разряд. Для выявления переполнения разрядной сетки существенное значение имеют значения переносов из значащей части числа в знаковый разряд **(C1)** и из знакового разряды за пределы разрядной сетки **(C2)**, который фиксируется регистре флагов **CF** (рис. 9).

Если C1 = C2, переполнения нет, в знаковом разряде разрядной сетки находится знак результата, а содержимое CF игнорируется, флаг OF = 0.

Если C1 ≠ C2, произошло переполнение, в знаковом разряде разрядной сетки находится старший разряд значащей части результата (разряд переполнения p), знак результата вытеснен за пределы разрядной сетки и находится в CF, OF = 1 указывает на то, что произошло переполнение.

Таким образом, значение флага **OF** определяется суммой по модулю 2 (⊕) переносов **C1** и **C2**

$$\mathbf{OF = C1 \oplus C2.}$$

Рассмотрим возможные ситуации при сложении двух чисел $\mathbf{C = X + Y}$, представленных в дополнительном коде в разрядной сетке $\mathbf{n = 3}$, значащая часть числа расположена в разрядах 0, 1, а знак в разряде 2.

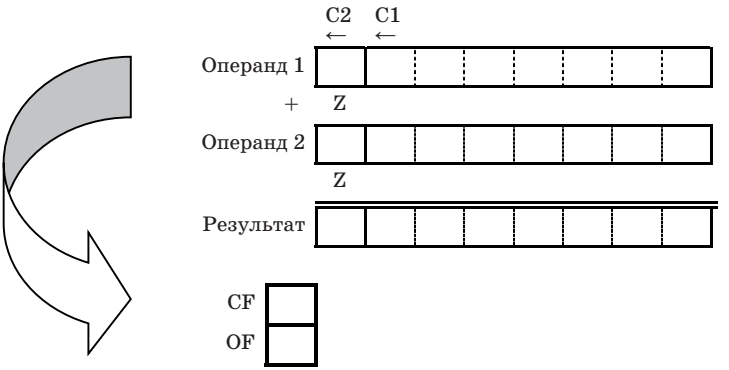


Рис. 9. Анализ переполнения разрядной сетки

1. $X_{10} = 2$, $X_2 = 10$, $+X_д = 0_z10$; $Y_{10} = 1$, $Y_2 = 01$, $+Y_д = 0_z01$;

	C2 ←	C1 ←	
X	0	1	0
+	Z		
Y	0	0	1
	Z		
C	0	1	1
	Z		

CF	0
OF	0

C1 = C2 =0, CF =0, переполнения нет, OF = 0, $C = 0_z11_2 = +3_{10}$

2. $X_{10} = 2$, $X_2 = 10$, $+X_д = 0_z10$; $Y_{10} = 3$, $Y_2 = 11$, $+Y_д = 0_z11$;

	C2 ←	C1 ←	
X	0	1	0
+	Z		
Y	0	1	1
	Z		
C	1	0	1
	p		Z

CF	0
OF	1

C1=1, C2=0, CF=0 (знак), переполнение, OF = 1, $C = 0_z101_2 = +5_{10}$

3. C = (–X) + (–Y); $X_{10} = 2$, $X_2 = 10$, $-X_д = 1_z10$; $Y_{10}=3$, $Y_2=11$, $-Y_д=1_z01$;

	C2 ←	C1 ←	
X	1	1	0
+	Z		
Y	1	0	1
	Z		
C	0	1	1
	p		Z

CF	1
OF	1

C1 = 0, C2 = 1, CF = 1 (знак), переполнение, OF = 1, $C_д = 1_z011_2$, $C_{пр} = 1_z101_2 = -5_{10}$

4. C = (–X) + (–Y); $X_{10} = 2$, $X_2 = 10$, $-X_д = 1_z10$; $Y_{10} = 1$, Y_2=01, $-Y_д = 1_z11$;

	C2 ←	C1 ←	
X	1	1	0
+	Z		
Y	1	1	1
	Z		
C	1	0	1
	Z		

CF	1
OF	0

эта «1» не учитывается в результате

C1 = 1, C2 = 1, CF = 1, переполнения нет, OF = 0, $C_д = 1_z01_2$, $C_{пр} = 1_z11_2 = -3_{10}$

7. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ. СЛОЖЕНИЕ ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

При сложении чисел с плавающей точкой необходимо учитывать, что мантисса представляется в прямом коде и в нормализованной форме, т. е. мантисса всегда имеет целую часть, равную 1, которая не хранится в разрядной сетке, а подразумевается. В поле порядка хранится характеристика, т. е. смещенный порядок.

Если порядки слагаемых одинаковы, то сложение выполняется по следующему алгоритму:

1. Мантиссы слагаемых складываются.
2. Порядок результата равен порядку слагаемых.
3. Мантисса нормализуется, при этом младшие разряды мантиссы, не вмещающиеся в ее разрядную сетку, отбрасываются. Если при нормализации мантиссы младшие разряды освобождаются, они заполняются незначащими нулями.

Если порядки слагаемых различны, перед выполнением перечисленных действий сначала выполняется выравнивание порядков – меньший порядок приводится к большему. Для этого мантисса сдвигается вправо с учетом целой части и на каждом шаге сдвига порядок увеличивается на «1» до тех пор, пока порядки слагаемых не станут равными.

Пример:

C = A + B; A = +1.111111100 E^{100111} (указан смещенный порядок)

B = +1.101100111 E^{100001}

Увеличиваем порядок числа **B** на 6, для чего сдвигаем мантиссу вправо на 6 разрядов. Теперь числа имеют одинаковый порядок $P_{см}$ = 100111, а мантисса числа **B** M_B = 0.000001101. При выравнивании порядков младшие шесть разрядов мантиссы числа **B** потеряны.

	Z	$P_{см}$						M								
A	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Z	$P_{см}$						M								
B	0	1	0	0	0	0	1	1	0	1	1	0	0	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рис. 10. Внутреннее представление чисел **A** *и* **B** *в разрядной сетке (целая часть мантисс равна 1 и не хранится)*

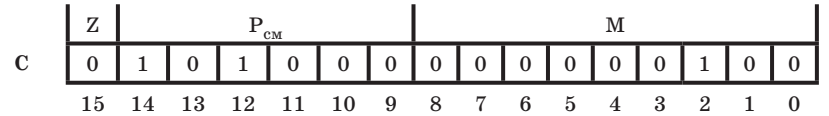


Рис. 11. Представление результата сложения **C = A + B** *в разрядной сетке*

Складываем мантиссы чисел

$$\mathbf{1.111111100 + 0.000001101 = 10.000001001.}$$

Мантисса результата не нормализована, поэтому сдвигаем ее вправо на один разряд, одновременно увеличивая порядок на «1».

Ответ: C = + **1.0000001001** $\mathbf{E^{101000}}$, представление результата в разрядной сетке показано на рис. 11.

Младшая единица мантиссы отброшена, так она не вмещается в разрядную сетку.

Умножение и деление чисел с плавающей точкой не требует выравнивания порядков. При выполнении этих операций мантиссы перемножаются (делятся), а порядки складываются или вычитаются. Однако, обязательным этапом операций умножения или деления является нормализация мантиссы полученного результата.

8. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ УМНОЖЕНИЯ В ВЫЧИСЛИТЕЛЬНОЙ МАШИНЕ

Умножение многоразрядных двоичных чисел в вычислительной машине является наиболее «трудоемкой» вычислительной операцией, занимающей значительное машинное время. Поэтому разрабатываются отдельные алгоритмы ускоренного умножения, реализуемые, как правило, с помощью специальных микропрограмм процессора, а в ряде случаев в структуру процессора включают специальные аппаратные блоки умножения. Мы рассмотрим только основные особенности выполнения умножения на примере простого алгоритма умножения целых **n** разрядных чисел без знака. Эти особенности связаны со следующим:

1) аппаратная реализация типового арифметико-логического устройства процессора предполагает непосредственное выполнение только сложения, логических операций и сдвигов, следовательно, алгоритм умножения должен быть сведен к последовательности выполнения только этих операций;

2) в современных процессорах все операции двуместные – отдельные команды процессора могут выполнять действия только над двумя операндами и, следовательно, для многоразрядных чисел алгоритм умножения должен быть основан на накоплении результата;

3) результат умножения **n** разрядных целых чисел без знака чисел имеет разрядность от 0 до **2 n**, т. е. при умножении возможно переполнение разрядной сетки на величину **n**.

4) в процессоре на уровне машинных команд предусмотрено выполнение умножения только целых чисел без знака (в некоторых процессорах знаковые в дополнительном коде, например, процессоры ARM), любые другие варианты реализуются отдельными библиотеками в различных системах программирования.

Переполнение разрядной сетки обрабатывается различными способами в зависимости от типов данных и особенностей решаемых задач.

1) При выполнении команды умножения операндов, соответствующих разрядности процессора, в процессоре результат размещается в двух регистрах. При этом часто слово переполнения не учитывается.

Например, при вычислениях в C/C++ выполнение следующего фрагмента программы

```
char var_a, var_b, var_c;
var_a = 25;
var_b = 15;
var_c = var_a * var_b;,
```

за счет отбрасывания старшего байта результата даст значение не 375, а *var _ c = 119*.

2) Умножение чисел в формате с фиксированной точкой исключит возможность переполнения, но результат будет округлен до значения, соответствующего размеру разрядной сетки.

3) При умножении чисел с плавающей точкой переполнения не возникает за счет усечения мантиссы и корректировки порядка.

Подчеркнем, что последние два случая в большей степени характерны для реализации библиотек математических функций, в которых учитываются особенности работы процессора.

Операция умножения двоичных чисел реализуется в вычислительной машине с применением операций сложения и сдвига. Возможные варианты базовых алгоритмов выполнения операции умножения представлены в табл. 3.

Характеристики эти алгоритмов мало отличаются, поэтому рассмотрим более подробно первый алгоритм, наиболее часто используемый на практике.

Введем некоторые обозначения.

Пусть необходимо вычислить произведение двух целых чисел без знака, представленных в разрядной сетке **n**

$$\mathbf{C = X_1 * X_2},$$

где $\mathbf{X_1}$ – *множимое*; $\mathbf{X_2}$ – *множитель*.

Таблица 3

Варианты алгоритмов умножения

Разряды множителя, с которого начинается умножение	Вариант умножения	Направление сдвига в процессе умножения		
		множимое	множитель	сумма частичных произведений
Младшие	1	Неподвижно	Вправо	Вправо
	2	Влево	Вправо	Неподвижно
Старшие	3	Неподвижно	Влево	Влево
	4	Вправо	Влево	Неподвижно

Для хранения промежуточных результатов вычисления и результата умножения, получаемого по окончании работы алгоритма, выделяются две ячейки памяти размером **n**, в которых будет храниться сумма частичных произведений **СЧП**, соответственно старшее и младшее слова $\mathbf{СЧП_H}$ и $\mathbf{СЧП_L}$ (индексы от слов high (высокий) и low (низкий) общеприняты в вычислительной технике). Таким образом, сумма частичных произведений имеет разрядность **2n**.

Обозначим $\mathbf{X_{20}}$ младший разряд множителя, сдвиг числа вправо на один разряд будем обозначать «→».

Сдвигу числа вправо соответствует операция деления числа на основание системы счисления. Механизм сдвига поясняется в разделе 11.3.3.

Возможный алгоритм умножения состоит из следующих шагов:

1. Сумма частичных произведений приравнивается к нулю

$$СЧП = 0.$$

2. Умножение множимого на младший разряд множителя с использованием таблицы умножения двоичных чисел (умножение на «0» дает «0», на «1» – множимое) и сложение результата с $\mathbf{СЧП_H}$

$$\mathbf{СЧП_H = СЧП_H + X_1 * X_{20}}.$$

3. Сдвиг **СЧП** вправо на один разряд

$$\mathbf{СЧП_H = СЧП_H \rightarrow}.$$

4. Сдвиг $\mathbf{X_2}$ вправо на один разряд

$$\mathbf{X_2 = X_2 \rightarrow}.$$

В результате повторения шагов 2 – 4 алгоритма **n** раз сумма частичных произведений будет равна $\mathbf{C = X_1 * X_2}$.

На этапах вычисления промежуточных значения СЧП возможны случаи переноса «1» за пределы разрядной сетки СЧП. Эта «1» запоминается в разряде CF регистра состояния и на шаге сдвига СЧП влево на один разряд возвращается в старший разряд разрядной сетки СЧП. Такой случай не является переполнением разрядной сетки.

Деление двоичных чисел производится аналогично.

Пример:

*	X_1	1	0	1	1				
	X_2	0	1	0	1				
		$СЧП_H$				$СЧП_L$				$\mathbf{X_2 = 0101 \leftarrow X_{20}}$
	+	0	0	0	0	0	0	0	0	
		1	0	1	1					$\mathbf{X_1 * X_{20}}$
		1	0	1	1	0	0	0	0	$\mathbf{СЧП_H + X_1 * X_{20}}$.
СЧП1	+	0	1	0	1	1	0	0	0	**СЧП →, X_2=0010**
		0	0	0	0					$\mathbf{X_1 * X_{20}}$
		0	1	0	1	1	0	0	0	
СЧП2	+	0	0	1	0	1	1	0	0	$\mathbf{X_2 = 0001}$
		1	0	1	1					
		1	1	0	1	1	1	0	0	
СЧП3		0	1	1	0	1	1	1	0	$\mathbf{X_2 = 0000}$
		0	0	0	0	0	0	0	0	
СЧП4		0	0	1	1	0	1	1	1	Результат

9. ДИАПАЗОН ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В РАЗЛИЧНЫХ ФОРМАТАХ ДЛЯ ДВОИЧНОЙ СИСТЕМЫ СЧИСЛЕНИЯ

При разработке программного обеспечения первоначально определяются типы данных, участвующих в вычислениях. Применительно к числовым данным определение типа задает размер разрядной сетки, ее формат, диапазон возможных значений и множество операций, применимых при обработке данных. Эти параметры данных программист должен тщательно планировать. Частые ошибки в программе связаны с выходом значения переменной за пределы диапазона возможных значений. Например, в программе на языке C/C++ в результате выполнения следующего фрагмента за счет выхода значения за пределы диапазона будет получено значение `var_a = 14464`:

```
unsigned short int var_a;
var_a = 80000;.
```

Приведем диапазоны представления чисел в рассмотренных выше форматах в разрядной сетке **n**.

1. Целое число без знака.

В языке C/ C++ может быть, на пример, объявлено

```
unsigned char var_a;
unsigned int var_a;
```

Наименьшее число состоит из всех «0», для максимального числа разрядная сетка заполнена «1», следовательно,

$$0 \le X \le 2^n - 1.$$

2. Целое число со знаком в прямом коде.

Учитывая, что один разряд отводится для знака, а, следовательно, значащая часть числа занимает **n–1** разряд, в ней хранится модуль числа, максимальное значение которого $2^{n-1} - 1$,

$$-(2^{n-1} - 1) \le X \le 2^{n-1} - 1.$$

Примеры объявления целого числа со знаком в языке C/ C++:

```
signed char var_a;
signed int var_a;
```

Обычно описатель `signed` опускается и эквивалентное объявление

```
char var_a;
int var_a;
```

приводит к такому же результату, однако это может вызвать трудно выявляемые ошибки в программе. Современные подходы к программированию для повышения читаемости программы и уменьшения

количества возможных ошибок рекомендуют по возможности избегать использования принципов умолчания в тексте программы.

3. Целое число в дополнительном коде.

Максимальное значение целого числа в дополнительном коде совпадает со значением в прямом коде, так как это число положительное и его представление совпадает с представлением в прямом коде.

Минимальное число – это наибольшее по модулю отрицательное число, записанное как дополнение до 2^n в виде $(2^n - |X|)$. Для максимального по модулю отрицательного числа значащая часть дополнения должна быть равна 0, следовательно, это число должно иметь 1 в старшем, знаковом, разряде, и нули в значащей части числа. Соответствующее значение в дополнительном коде, имеющее в значащей части дополнительного кода все «0», и «1» в знаковом разряде. Модуль этого числа равен 2^{n-1}, поэтому минимальное значения числа в дополнительном коде равно -2^{n-1}.

Таким образом, при использовании дополнительного кода,

$$-2^{n-1} \le X \le 2^{n-1} - 1,$$

т. е. наименьшее число по модулю на «1» больше, чем наибольшее. Представление в дополнительном коде не симметрично относительно «0», однако отсутствует характерное для прямого кода наличие двух нулей (со знаками «+» и «–»).

Следует отметить, что если для наименьшего числа в дополнительном коде найти модуль, а для этого необходимо вычислить его дополнение, получим

$$2^n - 2^{n-1} = 2^{n-1},$$

т. е. в знаковом разряде «1» переполнения и знак выходит за пределы разрядной сетки.

4. Диапазон представления правильной дроби можно получить из приведенных выше соотношений, полагая, что для целого числа точка, разделяющая целую и дробную части целого числа, расположена справа от младшего разряда. Для получения правильной дроби ее необходимо сдвинуть в позицию перед знаковым разрядом на **n–1** разряд, что соответствует делению числа на 2^{n-1}.

Тогда наименьшее значение правильной дроби в прямом коде

$$-(2^{n-1}-1)/\,2^{n-1} = -(1 - 2^{-(n-1)}),$$

а наибольшее $\quad +(2^{n-1}-1)/\,2^{n-1} = +(1 - 2^{-(n-1)}).$

Число может быть в диапазоне

$$-(1 - 2^{-(n-1)}) \le X \le (1 - 2^{-(n-1)}).$$

Аналогично для дополнительного кода правильная дробь находится в диапазоне

$$-1 \le X \le (1 - 2^{-(n-1)}).$$

Обратите внимание, что в этом случае в диапазон входит целое число –1.

5. Диапазон представления числа с плавающей точкой зависит от используемого формата и правил округления при представлении мантиссы.

Для рассмотренного выше представления по стандарту IEEE 754 нормализованная мантисса представляется в прямом коде и ее модуль его значения находится в пределах

$$1 \le M_н < 10_2 ,$$

целая часть мантиссы всегда равна «1», не хранится в разрядной сетке и ее значение подразумевается. Наибольшее по модулю значение дробной части нормализованной мантиссы $\left(1-2^{-(n_m-1)}\right)$, что соответствует заполнению единицами всего поля мантиссы разрядностью $\mathbf{n_m}$. Таким образом, с учетом целой части $\left|M_H\right|_{max} = 1+\left(1-2^{-(n_m-1)}\right) = 2-2^{-(n_m-1)}$.

Наибольшему значению числа с плавающей точкой соответствует наибольший положительный порядок. Смещенный порядок размером $\mathbf{n_p}$ имеет значащую часть размером $\mathbf{n_p\text{-}1}$, это целое число, старший бит которого косвенно указывает на его знак. Максимальное значение порядка равно $\left(2^{n_p-1}-1\right)$.

Тогда диапазон представления числа с плавающей точкой

$$-\left(2-2^{-(n_m-1)}\right)\cdot 2^{\left(2^{n_p-1}-1\right)} \le X \le \left(2-2^{-(n_m-1)}\right)\cdot 2^{\left(2^{n_p-1}-1\right)}.$$

10. ПРЕДСТАВЛЕНИЕ СИМВОЛЬНЫХ ДАННЫХ В КОМПЬЮТЕРНЫХ СИСТЕМАХ. КОДОВЫЕ ТАБЛИЦЫ

10.1. Основные принципы кодирования символов

Символьные данные в компьютерных системах имеют большое значение. Как было показано ранее, информация может заключаться в символах или их взаимном расположении. Любая система для обмена данными использует определенный алфавит, а передача данных в цифровых системах, основанных на двоичном кодировании любых данных, требует соответствующего кодирования символьных данных.

Необходимость в двоичном кодировании символов возникла в связи с широким внедрением стартстопных телеграфных аппаратов для обмена сообщениями, изобретенных в 1872 г. французским инженером Жаном Бодо. В этих аппаратах каждый символ передавался с помощью пяти элементной двоичной последовательности. Для телеграфного аппарата Бодо разработал пяти битовую кодировку символов. Код Бодо лег в основу международного телеграфного пятиэлементного кода ITA2 (в СССР – МТК-2), который затем был расширен до семи бит на символ. Особенностью этого кода является введение специальных управляющих символов, позволяющих изменять регистры и, соответственно, расширять набор используемых символов. Предусматривались управляющие символы для организации процесса передачи и управления оборудованием.

Первые электронные вычислительные машины оснащались телеграфными аппаратами, используемыми в качестве устройств ввода-вывода, поэтому применение методов кодирования символов, принятых в телеграфии, в этих машинах представлялось вполне естественным. Однако задачи передачи данных и реализации алгоритмов обработки символьных данных принципиально различны и использование для кодирования данных в компьютерных системах телеграфных кодов ограничивает возможности реализации эффективных алгоритмов обработки символьных данных.

Общая идея построения любой кодовой таблицы достаточно проста. Всем символам алфавита, используемого в системе, присваивается уникальный порядковый номер, называемый «код символа». Таким образом, для всего набора символов, используемых в системе, создается таблица соответствия символов и их представлению в виде двоичного числа. Эту таблицу принято называть «кодовой таблицей». Расстановка (порядок) символов в кодовой таблице в общем

случае может быть произвольной. Однако, в зависимости от решаемых задач, порядок символов в кодовой таблице должен отвечать определенным требованиям. По многим причинам эти требования далеко не всегда выполняются для различных систем кодирования символов и на практике используется большое количество различных кодовых таблиц. С точки зрения эффективности реализации алгоритмов обработки данных в компьютерных системах можно выделить следующие требования к построению кодовых таблиц.

1. Символы национального алфавита должны следовать в лексикографическом порядке, соответствующем принятому в конкретной стране расположению символов в алфавите. Это упрощает алгоритмы сравнения символов и сортировки символьных строк.

2. Символы прописных букв алфавита должны иметь меньший код, чем строчных, так как символьные строки часто рассматриваются как предложения и тогда прописную букву можно выделить простым сравнением кодов символов, как чисел.

3. Двоичные коды символов прописных и строчных букв национального алфавита должны отличаться одним специфическим битом. Это позволяет упростить возможность реализовать алгоритмы, использующие символы без учета верхнего и нижнего регистров, например, в поисковых системах, и обеспечить быструю замену регистра (прописные, строчные), используя логические операции.

4. Символы цифр от «0» до «9» должны располагаться в кодовой таблице последовательно и таким образом, чтобы выделение значения числа не требовало арифметических операций, а могло быть выполнено с помощью более быстрых логических операций.

5. Перечисленные выше требования имеют большое значение для построения эффективных алгоритмов проверки корректности ввода данных в компьютерных системах.

6. В кодовой таблице должны быть предусмотрены позиции для управляющих символов и символов-разделителей. При этом управляющие символы должны быть сгруппированы в одной области кодовой таблицы и иметь такой код, который позволяет их выделить и обработать с помощью наиболее быстрых логических операций.

7. Должна быть обеспечена совместимость кодовых таблиц, позволяющая использовать алгоритмы и программы, разработанные ранее, в более новых разработках, которые стали возможными в связи с использованием более современных аппаратных средств и развитием информационных технологий.

Здесь следует отметить, что перечисленные требования являются желаемыми и практически не выполняются в полной мере

в современных кодовых таблицах, что связано, в первую очередь, с проблемами совместимости систем различных поколений и многообразием используемых национальных алфавитов.

10.2. Кодовая таблица ASCII

Большинство современных систем основывается на кодовой таблице ASCII (American standard code for information interchange), разработанной и стандартизованной с США в 1963 г. В основу кодовой таблицы ASCII положен пятиэлементный телеграфный код ITA2 (в СССР – МТК-2), используемый в то время как международный стандарт для обмена телеграфными сообщениями. Кодовая таблица ASCII выделяет для каждого символа 7 бит двоичного кода. Обычно используется 8 бит, старший бит всегда равен «0». Таким образом, в этой таблице представлены 128 символов, имеющих коды от «0» до 127_{10}. Окончательный вид кодовой таблицы ASCII был опубликован в 1967 году и с тех пор кодировка первых 128 символов остается неизменной, эту часть кодовой таблицы обычно называют «базовой таблицей ASCII» или US ASCII.

В дальнейшем кодировка была расширена до 256 символов. В расширенной таблице ASCII добавляются символы, имеющие в коде «1» в старшем разряде. Дополнительная часть таблицы регламентируется различными стандартами, определяющими по существу «диалекты» таблицы ASCII, например кодовые таблицы **KOI8-R**, ISO 8859, CP836, CP1251 (Windows 1251) и другие.

Общим для расширенных кодовых таблиц ASCII является 8 битное кодирование символов и стандартное кодирование символов базовой части таблицы.

Базовая таблица ASCII (табл.4) определяет коды:

– управляющих символов;

– десяти символов десятичных цифр;

– прописных и строчных символов латинского алфавита, состоящего из 26 букв;

– знаков препинания и разделителей.

Принято нумеровать строки и столбцы кодовой таблицы шестнадцатеричными цифрами, указывающими значения старшего и младшего полубайтов кода символа. Так, например, код символа «F» в табл. 4 0x46. Символы базовой части таблицы ASCII всегда кодируются байтом, имеющим «0» в старшем разряде.

Кодовая таблица ASCII изначально разрабатывалась как унифицированная таблица для телетайпов, используемых в телеграфии.

Таблица 4

Базовая таблица ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	**NUL**	SOH	STX	ETX	EOT	ENQ	ACK	BEL	**BS**	HT	**LF**	VT	FF	**CR**	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	**ESC**	FS	GS	RS	US
2.	(пробел)	!	«	#	$	%	&	‘	(	)	*	+	,	–	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	**DEL**

Поэтому в нее включены управляющие символы, необходимые в телеграфной связи. Многие из этих символов в настоящее время утратили свое прямое назначение и могут использоваться в различных приложениях и аппаратуре в соответствии с внутренними соглашениями.

Особенности структуры таблицы ASCII:

– Управляющие символы (выделены серым цветом) занимают первые две строки, т. е. в коде этих символов старшие три бита равны 0, за исключением символа «DEL», эти символы не имеют общепринятого изображения;

– Коды символов десятичных цифр имеют в старшем полубайте значение «0011», а в младшем значение самого десятичного числа.

Таблица 5

Некоторые управляющие символы таблицы ASCII

Код символа	Обозначение	Английское название	Русское название	Назначение
00	NUL	NULL	Пустой символ	Часто используется для обозначения конца строки символов
08	BS	BACKSPACE	Возврат на шаг	Перемещает позицию печати на один символ назад. При вводе используется для стирания предшествующего символа («забой»)
09	HT	HORIZONTAL TABULATION	Горизонтальная табуляция	Перемещает позицию печати к следующей позиции горизонтальной табуляции
0A	LF	LINE FEED	Перевод строки	Перемещает позицию печати на одну строку вниз (без возврата каретки). Разделяет строки текстовых файлов в Unix-системах
0D	CR	CARRIAGE RETURN	Возврат каретки	Перемещает позицию печати в крайнее левое положение (исходно – без перевода на следующую строку)
1B	ESC	ESCAPE	Отмена	
7F	DEL	DELETE	Удаление (забой)	Обычно используется для удаления символа

Это позволяет получить значение числа из кода символа поразрядной конъюнкцией этого кода со значением маски «0x0F»;

– Символы латинского алфавита расставлены в лексикографическом порядке, символы прописных букв имеют меньший код, чем строчных.

– Коды символов прописных и строчных букв отличаются только одним битом в пятом разряде. Это позволяет при программировании игнорировать этот бит, если система не должна быть чувствительна к верхнему или нижнему регистрам или изменять регистр с помощью простых поразрядных логических операций.

В табл. 5 приведены некоторые управляющие символы ASCII, активно используемые при работе с персональным компьютером.

Символы **NUL** и **DEL** в телетайпах не вызывали никаких действий. Телетайпы работали с семидорожечной перфолентой, и символу **NUL** соответствовало отсутствие пробивки на перфоленте. Для удаления любого символа было достаточно заполнить всю дорожку единицами – пробивкой. Это объясняет положение символа **DEL** в кодовой таблице. Сейчас он часто используется при редактировании текста и в различных приложениях.

Таблица ASCII не является единственной, применяемой в вычислительных системах. В частности, в высокопроизводительных отказоустойчивых серверах (мэйнфреймах), используемых практически во всех поисковых системах, облачных технологиях и других критически важных системах, основной кодировкой является EBCDIC (Extended Binary Coded Decimal Interchange Code – расширенный двоично-десятичный код обмена информацией). Русифицированная версия этой кодовой таблицы – ДКОИ-8 («двоичный код обработки информации»).

10.3. Национальные восьмибитовые кодовые таблицы на основе ASCII

Адаптация вычислительной системы к использованию в конкретной языковой среде требует создания кодовых таблиц, включающих национальный алфавит. История развития вычислительной техники показывает множество примеров решения этой задачи. В частности, для русификации кодовой таблицы ЭВМ IBM/360 (в СССР выпускались аналогичные ЭВМ серии ЕС ЭВМ) использовалась кодировка символов кириллицы, построенная по следующему принципу. Символы кириллицы, совпадающие по изображению с символами латинского алфавита, представляются одинаковым кодом.

Например, символы «а», «в», «с» и подобные имеют одинаковое положение в кодовой таблице как для латинского, так и для русского алфавитов. Специфичные для русского алфавита символы, например, «ж», «ш», «й» и другие, занимают свободные позиции в кодовой таблице. Этот подход сохранился до настоящего времени в русскоязычных кодовых таблицах (ДКОИ-8), используемых в мэйнфреймах фирмы IBM, являющихся результатом развития вычислительных машин линии IBM/360.

Применительно к системам, использующим 8-битную кодировку ASCII, в настоящее время общепринято использовать дополнительную часть расширенной таблицы ASCII для размещения национального алфавита. При этом для обеспечения совместимости различных систем сохраняются кодовые позиции общепринятых символов, не связанных с национальным алфавитом.

Характерным примером такой кодовой таблицы является KOI8-R, которая была разработана для обмена телеграфными сообщениями и используется до настоящего времени многими телекоммуникационными системами на базе операционной системы UNIX. Первая половина кодовой таблицы KOI8-R полностью совпадает с базовой частью таблицы ASCII. В дополнительную часть таблицы KOI8-R (см. табл. 6) Входят общепринятые для международной таблицы символы (специальные символы и символы псевдографики) и символы русского алфавита. Обращает внимание тот факт, что символы кириллицы расставлены в полном не соответствии с приведенными выше принципами.

– Строчные буквы кириллицы имеют код, меньший, чем прописные.

– Нарушен лексикографический порядок расположения символов в таблице.

Такая расстановка символов кириллицы связана с особенностями работы телетайпов. В них передача выполняется асинхронно побайтно, начиная с младшего разряда. Вероятность искажения старших разрядов при передаче существенно выше. Ошибка в двоичной системе эквивалентна инверсии соответствующего разряда. Коды символов кириллицы расположены таким образом, чтобы, в случае ошибки в старшем разряде, по возможности получился код латинского символа соответствующего по значению символу кириллицы. Например, при передаче символа «Ф» и искажении старшего бита будет принят символ «f». Следовательно, текст можно будет прочитать. Изменение порядка следования строчных и прописных букв обусловлено теми же причинами. Телеграммы писались пропис-

Таблица 6

Дополнительная часть кодовой таблицы KOI8-R

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	─	│	┌	┐	└	┘	├	┤	┬	┴	┼	▀	▄	▉	▌	▐
9.	░	▒	▓	⌠	■	·	√	≈	≤	≥		⌡	°	²	·	÷
A.	═	║	╒	ё	╓	╔	╕	╖	╗	╘	╙	╚	╛	╜	╝	╞
B.	╟	╠	╡	Ё	╢	╣	╤	╥	╦	╧	╨	╩	╪	╫	╬	©
C.	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
D.	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
E.	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
F.	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	З	Ш	Э	Щ	Ч	Ъ

ными буквами, так как многие аппараты не обеспечивали печать строчных букв. Соответственно, вероятность искажения прописных букв была существенно выше.

Использование кода KOI8-R в современных системах телекоммуникаций, использующих ОС UNIX, позволяет обеспечить совместимость аппаратуры и программного обеспечения, разработанных в разные годы. В этих системах задачи обработки текстовых массивов встречаются весьма редко и лексикографический порядок символов обычно не существенен.

Известно множество национальных кодовых таблиц, учитывающих различные аспекты архитектуры компьютеров и программного обеспечения. Большинство из них потеряли свое значение и не используются.

Применительно к современным системам имеют значение 8-битовые кодовые таблицы ISO 8859 и Windows 1251.

ISO 8859 – семейство ASCII-совместимых кодовых страниц, принятое в качестве международного стандарта. ISO – это Международная организация по стандартизации (International Organization for Standardization (англ.)). Стандарт ISO 8859 определяет набор кодовых страниц, соответствующих определенным национальным алфавитам, и задающих дополнительную часть расширенной таблицы ASCII. Эти страницы обозначаются: ISO 8859-1 (Latin-1), ISO 8859-2 (Latin-2) и т. д. Страница ISO 8859-5 (Latin/Cyrillic) кириллица, включающая символы кириллицы, приведена в табл. 9. Для русского языка эта страница соответствует ГОСТ [7].

Кодировка ISO 8859 применяется, в основном, в системах, основанных на ОС UNIX и кодирования веб-страниц, так как большинство веб-серверов используют эту операционную систему.

В приведенной таблице выделены управляющие символы, используемые для передачи и обработки данных в компьютерных системах и сетях. Эти символы повторяются практически в каждой кодовой странице ISO 8859 . Как следует из таблицы, для букв русского алфавита, кроме буквы «Ё», выдерживается лексикографический порядок.

Кодовая страница Windows 1251, используемая в операционных системах Windows приведена в табл.1 0. За счет исключения дополнительных управляющих символов в кодовую страницу Windows 1251

Таблица 9

Кодовая страница ISO 8859-5

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9.	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
A.		Ё	Ђ	Ѓ	Є	S	I	Ї	J	Љ	Њ	Ћ	Ќ	SHY	Ў	Џ
B.	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
C.	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
D.	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
E.	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F.	№	ё	ђ	ѓ	є	s	і	ї	j	љ	њ	ћ	ќ	§	ў	џ

Таблица 10

Кодовая страница Windows 1251

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	Ђ	Ѓ	‚	ѓ	„	…	†	‡	€	‰	Љ	‹	Њ	Ќ	Ћ	Џ
9.	ђ	‘	’	“	”	•	–	—		™	љ	›	њ	ќ	ћ	џ
A.		Ў	ў	J	¤	Ґ	¦	§	Ё	©	Є	«	¬		®	Ї
B.	°	±	І	і	ґ	µ	¶	·	ё	№	є	«	j	S	s	ї
C.	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
D.	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
E.	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
F.	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

включены практически все символы, используемые для форматирования русскоязычного текста, что выгодно отличает эту кодовую страницу от страницы ISO 8859-5. Обе кодовые страницы содержат все символы украинского, белорусского, сербского, македонского, болгарского и русского языков.

В заключение следует отметить, что возможности 8 битных кодовых таблиц на сегодняшний день практически полностью исчерпаны. Серьезные проблемы использования этого формата возникают при реализации систем, использующих национальный алфавит с большим количеством символов, например, китайский, японский и многие другие.

10.4. Кодирование символов в Unicode (Юникод)

Рассмотренных выше 8 битовых кодировок символов в целом достаточно при работе автономной компьютера в конкретной языковой среде. Переход на глобальные сети, создание единого информационного пространства, использующего ресурсы множества распределенных компьютеров различной архитектуры, требует использования единого подхода к цифровому представлению всех возможных символов. Такой подход реализован в стандарте кодирования символов UNICODE (Юникод), в настоящее время преобладающем в Интернете и ряде операционных систем. Стандарт предложен в 1991 г. некоммерческой организацией «Консорциум Юникода» и практически не имеет ограничений на количество закодированных символов. Принятые соглашения позволяют закодировать 1 112 064, хотя принципиально количество символов может быть увеличено до 2^{31} (2 147 483 648).

Стандарт UNICODE разделен на две части:

– Универсальный набор символов (Universal character set, **UCS**), в котором перечисляются все допустимые по стандарту символы и каждому символу определена кодовая ячейка размером, в зависимости от версии, 16 или 32 бита. Код символа записывается как целое число без знака в шестнадцатеричной форме с префиксом U+, например, U+04FF.

– Семейство кодировок (Unicode transformation format, **UTF**) задает алгоритмы преобразования кодов символов при передаче в поток или файл.

В UNICODE коды разделены на несколько областей. Область U+0000 до U+00FF отведена для расширенной таблицы ASCII. Причем в дополнительной части расположены, в том числе, дополни-

тельные управляющие символы и символы форматирования, определенные стандартом ISO 8859. Они занимают диапазон от U+0080 до U+00AF. Остальная часть **UCS** включает только графические символы, имеющие видимое изображение:

– буквы, содержащиеся хотя бы в одном из обслуживаемых алфавитов;

– цифры;

– знаки пунктуации;

– специальные знаки (математические, технические, идеограммы и пр.);

– разделители.

Все кодовое пространство универсального набора символов разбито на 17 плоскостей по 2^{16} – (65536) символов. Нулевая плоскость называется базовой и содержит символы наиболее употребительных письменностей. Остальные плоскости включают менее употребительные символы, например, математические, нотные и даже символы древних, вымерших, языков. Таким образом, при использовании UNICODE полностью отпадает необходимость переключения кодовых страниц.

Семейство кодировок UTF определяют способ представления символов UNICODE для хранения и передачи данных. Используются способы UTF-8, UTF-16 и UTF-32. Подробное рассмотрение этих способов выходит за пределы настоящего пособия и требуется только в частных случаях разработки специальных приложений. В современных системах программирования их реализации включены в функциональные возможности системы. Поэтому рассмотрим только основные особенности этих семейств кодировок.

UTF-8 представляет собой представление символов UNICODE, для которого используется от 1 до 4 байт. Символы представлены словами переменной длинны. В качестве признака длинны кода символа используются старшие биты кода UNICODE. Так, например, символы, имеющие код в диапазоне от U+0000 до U+007F закодированы одним байтом и этот код полностью совпадает с базовой частью таблицы ASCII. В этом случае внутренние представления символьной строки в коде ASCII и UTF-8 полностью совпадают и это обеспечивает совместимость в работе систем различных поколений. Текст, использующий только символы базовой части таблицы ASCII, в такой кодировке является наиболее компактным и воспринимается любой системой.

Двумя байтами в UTF-8 кодируется (2^{12}–128) символов, имеющих коды UNICODE от U+0080 до U+07FF. В этой области находятся

наиболее часто употребляемые символы национальных алфавитов, в том числе символы кириллицы.

Описанные свойства UTF-8 обусловили ее преимущественное распространение в сети Интернет и во многих современных приложениях, в том числе в UNIX системах.

Основной недостаток UTF-8 заключается в весьма сложном алгоритме декодирования, что в ряде случаев приводит к ощутимой потере быстродействия системы из-за того, что в ряде алгоритмов требуется последовательный просмотр и, соответственно декодирование всего текстового массива.

В кодировании UTF-16 также используется переменная длина кода для представления символов. Это может быть 2 или 4 байта. В некоторых случаях этот метод кодирования оказывается более гибким, чем UTF-8. UTF-16 часто используется в отдельных приложениях, например, в Java, внутреннее представление символов в операционной системе WINDOWS. По сравнению с UTF-8, UTF-16 обладает большей избыточностью, но позволяет представить практически все мировые системы письма двухбайтным кодом символа. Однако, UTF-16 не обеспечивает совместимость с восьмиразрядными кодовыми таблицами ASCII, что не является ограничением в рамках конкретных операционных систем или приложений, но может привести к определенным проблемам при организации сетевого взаимодействия компьютерных систем.

В UTF-32 каждый символ представляется 4 байтным двоичным числом в соответствии с кодовой таблицей UNICODE.

Таким образом, использование UNICODE в современном программном обеспечении позволяет решить многие проблемы с представлением и обработкой символьных данных и обеспечить взаимодействие различных систем, при этом внутренние формы хранения символьных данных в рамках отдельного приложения может быть произвольным.

11. ОСОБЕННОСТИ ПРЕДСТАВЛЕНИЯ ДАННЫХ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С/С++

Организация использования рассмотренных выше типов данных имеет свои особенности в различных языках программирования. В частности, в языке программирования С/С++ основными типами данных являются целое со знаком, целое без знака и числа с плавающей точкой, для хранения которых выделяется память, соответствующая конкретному типу. В языке С/С++ обязательным является явное объявление данных. Объявление данных определяет тип данного и его имя (идентификатор).

Язык С++ является продолжением идеологии С применительно к технологии объектно-ориентированного программирования, и унаследовал его синтаксис, как и многие базовые структуры. Поэтому в учебном пособии принято обозначение «С/С++», а излагаемый материал применим в обоих языках программирования. Правила реализации различных типов данных и допустимых для них операций устанавливаются международным стандартом языка. На сегодняшний день действует стандарт языка С ISO/IEC 9899:201x [9], короткое название которого С11. Для языка С++ действует стандарт ISO/IEC 14882:2014(E) Programming Language C++ [10], или С++14. Некоторые особенности типов данных специфичны для конкретной реализации языка, используемого компилятора. В данном пособии рассматривается реализация Microsoft языка С/С++, компилятор Microsoft Visual C++ 2017.

11.1. Простейшие типы данных и их внутреннее представление

Основными простейшими типами данных в языке С/С++ являются данные целого типа (`int`, `char`, `bool`) и данные с плавающей точкой (`float` и `double`). Типы данных `char` и `int` могут быть дополнены описателями `signed` или `unsigned`, определяющие наличие или отсутствие поля для знака в двоичном представлении числа. Типы данных `float` и `double` предназначены для хранения чисел с плавающей точкой и отличаются количеством выделяемой памяти, и соответственно, шириной полей для записи мантиссы и порядка.

11.1.1. Данные целого типа `int`

Базовым типом данных для представления целых чисел в языке С/С++ является тип `int`. Размер данного типа `int` зависит от раз-

рядности процессора и используемого компилятора. Отличительной особенностью языка C/C++ является то, что объявление данных этого типа не дает строгих ограничений на операции, которые могут к ним применяться, и соответственно, в зависимости от особенностей алгоритма, эти данные могут использоваться как числовые, символьные или логические.

Для хранения данного целого типа [11] может быть выделено 2, 4 или 8 байт. Размер выделяемой памяти задается спецификатором `short`, `long` или `long long`. При использовании короткого типа `short int` или длинных типов `long int` и `long long int` слово `int` можно не указывать. Целочисленные данные в C/C++ могут иметь знаковую или беззнаковую форму в зависимости от наличия соответствующего описателя (`signed` или `unsigned`). Так, целочисленные данные с соответствующими описателями и спецификаторами отличаются размером выделяемой для хранения памяти, формой представления данных и, соответственно, диапазонами принимаемых значений. Например:

```
signed short int var_a = - 5;
```

При объявлении переменной будет выделено 2 байта памяти, старший бит будет использоваться для указания знака числа, число хранится в дополнительном коде.

```
unsigned int var_a = 352568;
```

При объявлении переменной будет выделено 4 байта памяти, все биты соответствуют значащей части числа, число хранится в прямом коде.

```
signed long int var_a = 123654279;
```

При объявлении переменной будет выделено 4 байта памяти, старший бит будет использоваться для указания знака числа, число хранится в дополнительном коде.

В языке C/C++ данные целого типа могут задаваться в десятичной, восьмеричной и шестнадцатеричной системах счисления. Некоторые компиляторы поддерживают и двоичную форму записи чисел в программе. Десятичное число указывается в виде последовательности цифр от 0 до 9:

```
signed int a_dec = 6882;
```

Любое целое число, начинающееся с 0, после которого следуют цифры от 0 до 7, рассматривается компилятором как заданное в восьмеричной системе счисления:

```
signed int a_oct = 015342;
```

Признаком числа в шестнадцатеричной системе счисления является 0x (0X) в качестве первых его символов. Запись числа при

этом может содержать цифры от 0 до 9, а также символы A, B, C, D, E, F (могут использоваться как прописные, так и строчные буквы):

```
signed int a_hex = 0x1AE2;
```

Размер памяти, выделяемой для хранения целого числа с различными спецификаторами, зависит от используемого компилятора и разрядности машины. Однако, в стандарте C11 для всех целых типов предусмотрено:

```
1 == sizeof(char) <= sizeof(short int) <= sizeof(int) <=
sizeof(long int) <= sizeof(long long int)
```

Функция `sizeof()` возвращает количество байт для хранения аргумента. В качестве аргумента могут использоваться ключевое слово, обозначающее тип данных, переменная, массив, структура и т.п.

Язык C/C++ поддерживает и другие целочисленные типы, позволяющие вне зависимости от компилятора установить размер памяти, выделяемой для хранения числа. Это типы `_int8`, `_int16`, `_int32`, `_int64`. Они также могут быть представлены в знаковой или беззнаковой форме при использовании описателей `signed` или `unsigned` соответственно.

11.1.2. Данные типа `char`

Данные типа `char` занимают в памяти 1 байт, в котором может храниться целое число как со знаком, так и без знака. Чаще всего этот тип данных используется для записи кода символа из используемой в системе 8-ми битной таблицы кодировки. В программе на языке C/C++ для записи символа в ячейку памяти, объявленную как `char`, возможно указание единичного символа в одинарных кавычках или непосредственно кода символа, записанного как целое число. Например:

```
unsigned char A_symbol = 'A';    // латинская буква A
или
unsigned char A_symbol = 0x41; // код символа 'A'
                          // в 16-ричной системе счисления
```

Группа символов, начинающихся со специального символа обратной косой черты '\', интерпретируются компилятором особым образом (табл. 11). В табл. 11 первые 7 символов представляют собой управляющие символы, или так называемые escape-последовательности, предназначенные для выполнения различных действий на устройствах отображения.

Таблица 11

Управляющие и специальные символы в языке C/C++

№	Символ	Код символа в ASCII	Описание
1	'\a'	0x07	(alert) Создает звуковое или видимое оповещение
2	'\b'	0x08	(backspace) Перемещает курсор в предыдущую позицию в текущей строке. Если курсор находится в начальной позиции строки, поведение устройства отображения не определено
3	'\f'	0x0C	(подача страницы) Перемещает курсор в начальную позицию в начале следующей логической страницы
4	'\n'	0x0A	(новая строка) Перемещает курсор в начальную позицию следующей строки
5	'\r'	0x0D	(возврат каретки) Перемещает курсор в начальную позицию текущей строки
6	'\t'	0x09	(горизонтальная табуляция) Перемещает курсор на следующую позицию горизонтальной табуляции в текущей строке
7	'\v'	0x0B	(вертикальная табуляция) Перемещает курсор в начальную позицию следующей позиции вертикальной табуляции.
8	'\\'	0x5C	Символ обратной косой черты
9	'\''	0x27	Символ кавычки
10	'\?'	0x3F	Символ вопросительного знака
11	'\0*восьмиричное_число*'		Явное задание кода символа в восьмеричной системе счисления
12	'\0x*16-тиричное_число*'		Явное задание кода символа в 16-ричной системе счисления
13	'\0'	0x00	Нуль-символ (нулевое значение байта)

При использовании типа char для хранения целого числа, данные могут рассматриваться компилятором как числа без знака – unsigned char или как числа со знаком – signed char, обычно по умолчанию signed можно опустить. Если тип char рассматривается как signed, то старший бит его кода определяет знак и, соответственно, диапазон принимаемых значений – от –128 до +127. В случае беззнаковой формы unsigned char все биты рассматриваются как код числа, а диапазон значений – от 0 до 255. Например:

```
signed char  var_a = -5;
signed char  var_b = 50;
signed char  var_c = var_a + var_b;
```

11.1.3. Логический тип bool

Тип bool согласно стандарту C11 [9] представляет собой беззнаковый целый тип данных, которые могут принимать значения 1 (true, «истина») или 0 (false, «ложь»). В языке C/C++ для хранения данного типа bool используется 1 байт (8 бит) памяти, т. е. теоретически может храниться 256 значений. Однако, при попытке присвоить в переменную, объявленную как bool, любого значения, отличного от нуля, произойдет автоматическое преобразование данного в true и хранится будет единица. Это позволяет предотвратить ошибки при использовании логических данных в выражениях. Так программа:

```
// Пример программы с использованием переменной типа bool в выражении
#include <stdio.h>
int main() {
     bool bool_var = 55;
     signed int var_a = 3 + bool_var;
     printf("bool_var = %d\n", bool_var);
     printf("var_a = %d\n", var_a);
     return 0;
}
```

выведет на экран результат:

```
bool_var = 1
var_a = 4
```

11.1.4. Данные с плавающей точкой

В языке C/C++ используется стандарт IEEE-754 для представления чисел с плавающей точкой. В Microsoft Visual C++ [11] тип float

занимает 4 байта, из которых один разряд выделяется под знак, 8 разрядов – под поле порядка и 23 разряда – под поле мантиссы. Представление `long double` и `double` идентичны и занимают в памяти по 8 байт, для хранения порядка выделяется 11 разрядов, а для мантиссы – 52 разряда. Однако типы `long double` и `double` рассматриваются компилятором как разные типы. Типы данных с плавающей точкой всегда являются знаковыми.

Особенностью арифметики с плавающей точкой в языке C/C++ является то, что в целях повышения точности, в выражениях при вычислениях операнды типа `float` автоматически преобразуются компилятором в тип `double`. Это оправдывает использование типа `float` только для экономии памяти при хранении данных, но может привести к появлению ошибок (см. пример в разделе 5.4). Более того, при записи дробной части числа в формате с плавающей точкой хранится округленное значение, что следует учитывать при написании программ, и в особенности при использовании операций сравнения. Не рекомендуется для чисел с плавающей точкой использовать сравнение на равенство (с помощью оператора «==»). Взамен этого, например, можно задать некоторую точность вычислений и использовать логические операции для того, что бы установить, достаточно ли близки два числа. В качестве примера приведем программу, где в цикле к переменной `var_a`, инициализированной нулем, прибавляется 0.1 и цикл имеет 10 итераций. Ожидается, что в итоге в переменной `var_a` должно храниться число 1. Далее приведен фрагмент программы, демонстрирующий накопление погрешности при вычислениях с использованием чисел с плавающей точкой.

```
double var_a=0;
for (int i = 0; i < 10; i++){
                var_a += 0.1;
        printf("var_a = %.16f\n", var_a);
}
printf("При сравнении на равно: \n");
if (var_a == 1.) printf("var_a == 1 \n");
else printf("var_a ~= 1 \n");
printf("При сравнении с заданной погрешностью: \n");
if (abs(var_a - 1.) <= DBL_EPSILON) printf("var_a == 1 \n");
else printf("var_a ~= 1 \n");
```

В приведенном фрагменте программы `DBL_EPSILON` – это встроенная константа, равная погрешности вычислений между данными

типа double. В результате вычислений накапливается ошибка, поэтому после работы цикла переменная a лишь приближенно равна 1.

```
var_a = 0,1000000000000000
var_a = 0,2000000000000000
var_a = 0,3000000000000000
var_a = 0,4000000000000000
var_a = 0,5000000000000000
var_a = 0,6000000000000000
var_a = 0,7000000000000000
var_a = 0,7999999999999999
var_a = 0,8999999999999999
var_a = 0,9999999999999999
При сравнении на равно:
var_a ~= 1
При сравнении с заданной погрешностью:
var_a == 1
```

В языке C/C++ размер памяти, выделяемой системой для заданного типа данных, можно узнать, используя функцию sizeof(). Максимальное и минимальное значение данных различных типов встроены в язык как именованные константы (заголовочный файл стандартной библиотеки общего назначения языка <limits.h>).

Пример. С помощью использования встроенных констант и функции sizeof() составить таблицу, отражающую простейшие типы данных в языке C/C++ с указанием выделяемой для их хранения памяти и диапазоном значений, принимаемых данными каждого из указанных типов.

```
// Программа, выводящая на экран в виде таблицы простейшие
// типы данных, их размер и диапазон значений

#include <iostream>
using namespace std;

int main(){
setlocale(LC_ALL, "rus");
// вывод таблицы с использованием строенных в язык констант
printf("   РАЗМЕР ТИПОВ ДАННЫХ И ДИАПАЗОН ПРИНИМАЕМЫХ ЗНАЧЕНИЙ    n");
printf("/-------------------------------------------------------\\\n");
printf("|    Тип данных    |Кол-во байт|Мин.значение|Макс. значение|\n");
printf("|-------------------------------------------------------\n");
printf("|      bool       |%11d|%12d|%14d|\n", sizeof(bool), false, true);
printf("|     char        |%11d|%12d|%14d|\n", sizeof(char), CHAR_MIN,
                                                          CHAR_MAX);
```

```
printf("|  unsigned char   |%11d|%12d|%14d|\n", sizeof(char), 0,
                                                          UCHAR_MAX);
printf("|    short int     |%11d|%12d|%14d|\n", sizeof(short int),
                                                  SHRT_MIN, SHRT_MAX);
printf("|unsigned short int|%11d|%12d|%14u|\n",sizeof(unsigned short
                                                   int), 0, USHRT_MAX);
printf("|        int       |%11d|%12d|%14d|\n", sizeof(int), INT_MIN,
                                                     INT_MAX);
printf("|  unsigned int    |%11d|%12d|%14u|\n", sizeof(unsigned int), 0,
                                                          UINT_MAX);
printf("|     long int     |%11d|%12d|%14d|\n", sizeof(long int),
                                                  LONG_MIN, LONG_MAX);
printf("|unsigned long int |%11d|%12d|%14u|\n",sizeof(unsigned
                                               long int),0,ULONG_MAX);
printf("|      float       |%11d|%12.2e|%14.2e|\n", sizeof(float),
                                                    FLT_MIN, FLT_MAX);
printf("|     double       |%11d|%12.2e|%14.2e|\n",sizeof(double),
                                                     DBL_MIN,DBL_MAX);
printf("|   long double    |%11d|%12.2e|%14.2e|\n",sizeof(long double),
LDBL_MIN, LDBL_MAX);
printf("\\-----------------------------------------------------------/\n");

system("pause");
return 0;
}
```

Результат работы программы:

```
   РАЗМЕР ТИПОВ ДАННЫХ И ДИАПАЗОН ПРИНИМАЕМЫХ ЗНАЧЕНИЙ
/-------------------------------------------------------------\
|    Тип данных    |Кол-во байт|Мин.значение|Макс. значение  |
---------------------------------------------------------------
|       bool       |          1|           0|               1|
|       char       |          1|        -128|             127|
|  unsigned char   |          1|           0|             255|
|    short int     |          2|      -32768|           32767|
|unsigned short int|          2|           0|           65535|
|        int       |          4| -2147483648|      2147483647|
|  unsigned int    |          4|           0|      4294967295|
|     long int     |          4| -2147483648|      2147483647|
|unsigned long int |          4|           0|      4294967295|
|      float       |          4|    1,18e-38|        3,40e+38|
|     double       |          8|   2,23e-308|       1,80e+308|
|   long double    |          8|   2,23e-308|       1,80e+308|
\-------------------------------------------------------------/
```

11.2. Идентификаторы. Переменные. Константы

В процессе написания программы на языке высокого уровня используются различные объекты, типы данных, файлы и т.п., которые обозначаются не двоичными кодами, соответствующие внутреннему машинному представлению, а понятными человеку именами. Эти имена общепринято называть идентификаторами.

11.2.1. Идентификаторы

Идентификатор – это символическое имя, которое определяет любой объект в программе. Этими объектами могут быть переменные, используемые в программе, функции, файлы, и т. д. Символьное обозначение объектов через идентификаторы обеспечивает читаемость и переносимость кода.

В языке C/C++ идентификатор, или имя, может состоять из прописных и строчных букв латинского алфавита, цифр и знака нижнего подчеркивания, однако первым символом идентификатора должна быть буква или знак подчеркивания. Язык C/C++ рассматривает буквы верхнего и нижнего регистров как различные символы. Идентификатор не может совпадать с ключевыми словами языка (ключевые слова – это слова, зарезервированные системой только для использования компилятором C/C++, в современных средах разработки ключевые слова подсвечиваются отличным от основного текста программы цветом). Крайне нежелательно создавать идентификаторы, совпадающие с именами функций, уже содержащихся в библиотеках C/C++.

Применительно к данным идентификатор позволяет при разработке программного кода связать данные в памяти и особенности выполнения операций над этими данными. Основными данными в C/C++ являются переменные и константы.

11.2.2. Переменные

Переменная – это именованная область памяти, в которой хранятся данные. Данные, находящиеся в переменной, называют значением этой переменной. В процессе выполнения программы значение переменной может изменяться. В языке C/C++ для объявления переменной используется следующий синтаксис:

```
тип_данных_переменной  имя_переменной;
```

`тип_данных_переменной` определяет объем памяти, который необходимо выделить для хранения данных, форму представления дан-

ных и те операции, которые над этими данными можно выполнять. `имя_переменной` представляет собой идентификатор переменной.

Так, строка программного кода:

```
signed int  var_a;
```

означает, что в памяти необходимо выделить 4 байта (в соответствии с размером `int`), обращение к этой ячейке будет выполняться по адресу, имеющему символическое имя `var_a`, старший бит использовать для указания знака числа, число хранится в дополнительном коде.

Для присваивания значения переменной в языке C/C++ используется оператор «=». Операция присваивания несет в себе следующий смысл: в ячейку с адресом, ассоциированным с идентификатором, указанным слева от оператора «=», скопировать данное, указанное справа от оператора «=».

Задать начальное значение переменной с помощью оператора присваивания можно при объявлении, а также после него. Например:

```
signed int  var_a = 2;
```

В результате в ячейке, выделенной для `var_a`, будет записано число 2. Такая запись носит название определение переменной или объявление с инициализацией.

Если при объявлении переменной не выполняется ее инициализация, по адресу `var_a` будут доступны данные, хранившиеся ранее в выделенных ячейках и являющиеся по сути «мусором». Неинициализированные переменные могут содержать любое значение, и их применение приводит к неопределенному поведению программы.

11.2.3. Константы. Именованные константы

При разработке программ зачастую используют данные, значения которых не изменяется в процессе выполнения программы. Такие данные называют константами. Это могут быть, к примеру, определённые в математике постоянные, как число π, размерность массива и другие. Все константы можно разделить на две группы – именованные константы, и константы, не имеющие имени и записываемые в программе через свои текстовые представления (литералы). Хорошим стилем программирования считается использование именованных констант и сведение к минимуму использования неименованных констант. Именованная константа имеет символическое имя – идентификатор, по которому к ней ведется обращение в тексте программы.

Синтаксис определения именованной константы совпадает с синтаксисом определения переменной, но перед указанием типа кон-

станты ставится ключевое слово const. Компилятор языка следит за тем, чтобы данные, указанные как const, не были изменены в программе. Например:

```
const float PI = 3.14159265358979323846;
const unsigned int MAX_VAL = 1000;
```

Принято идентификаторы именованных констант составлять из букв верхнего регистра, объединяя слова символом нижнего подчеркивания, чтобы отличать их от имен переменных и функций. В случае изменения значения именованной константы достаточно изменить только одну строчку программного кода – определение константы, и значение поменяется во всех местах ее использования.

В языке С имеется еще один способ определения символического имени константы – использование директивы препроцессора #define. Директива #define обычно используется для замены часто используемых в программе констант, ключевых слов, операторов и выражений осмысленными идентификаторами. Первое слово, стоящее после #define, обозначает идентификатор, а следующее, указанное через пробел, – значение константы. Например:

```
#define PI 3.14159265358979323846
#define MAX_VAL 1000
```

При компиляции указанное значение подставляется вместо идентификатора во все места его использования.

Именованные константы, определенные через ключевое слово const или через директиву препроцессора #define, так же, как и переменные, могут участвовать в выражениях и операциях, однако их значения не могут быть изменены в процессе выполнения программы.

11.2.4. Неименованные константы

Неименованные константы используются в операциях и выражениях, а в тексте программы они представлены текстовой записью их значения (литералом) [12]. Компилятор переводит текстовую запись константы в ее двоичное представление. Константа может быть числовой: 2, 3.5e5, 34567.455; логической: true, false; символьной: 'A'; строковой: "Hello, world!".

Тип и размер константы определяется компилятором автоматически на основе ее синтаксиса в тексте программы. Программист с помощью определенных префиксов и суффиксов может явно указать компилятору, к какому типу нужно привести значение неименованной константы.

Таблица 12

Основные литералы в языке C/C++

Синтаксис	Пояснение	Пример
число	Целое в десятичной системе счисления, для записи числа используются цифры 0-9	10 526 -15
0*восьмиричное_число*	Целое в восьмеричной системе счисления, для записи числа используются цифры 0-7	010 // (8_{10}) 01016 // (526_{10}) -017 // (-15_{10})
0x*16-тиричное_число* 0X*16-тиричное_число*	Целое в 16-ричной системе счисления, для записи числа используются цифры 0-9, латинские буквы A,B,C,D,E,F	0x10 // (16_{10}) 0x20E // (526_{10}) -0xF // (-15_{10})
0b*двоичные_число* 0B*двоичные_число*	Целое в двоичной системе счисления, для записи числа используются цифры 0 и 1. Реализовано не во всех версиях языка	0b10 //(2_{10}) 0b1000001110 //(526_{10}) -0b1111 //(-15_{10})
*число*L *число*l	Хранение числа в формате long int	-10L // *десятичное число в формате long* 043L // *восьмеричное число в формате long* 0xF6385L // *16-ричное число в формате long* 6.56L // *вещественное число в формате long double*

Окончание табл. 12

Синтаксис	Пояснение	Пример
*число*U *число*u	Хранение целого числа в беззнаковой форме	54U // десятичное число в формате unsigned int 0743U //*восьмеричное число в формате unsigned int* 0xF6LU // *16-ричное число в формате unsigned long int*
целая_часть.дробная_часть	Вещественное число в форме записи с десятичной точкой	52.365 -0.26
*целая_часть.дробная_часть*E*степень* *целая_часть.дробная_часть*e*степень*	Вещественное числа в научной форме записи. Под E (e) подразумевается основание системы счисления	5.2365E1 -26E-2
*число*F *число*f	Хранение вещественного числа в формате float	4.5F
true false	Константы типа bool. Вместо записи «true» допускается запись «1» (или иное ненулевое значение), вместо «false» – «0»	
'*символ*'	Символ, заключенный в одиночные кавычки представляет символьную константу	'A' 't' '\n'
"*строка_символов*"	Символы (один или несколько), заключенные в двойные кавычки, представляют строковую константу. В конце автоматически добавляется нуль-символ	"Hello, world!"

Целые константы по умолчанию хранятся в формате `signed int`. В случае, если размера `int` недостаточно для хранения значения константы, для ее записи выбирается тип `signed long int` или `unsigned long int`. Вещественные константы хранятся по умолчанию в формате `double`. Символьные константы имеют тип `char`, а строковые – представляют собой массивы из элементов типа `char`.

Примеры основных литералов, их суффиксов и префиксов в языке C/C++ приведены в табл. 12.

11.3. Выражения и операции в языке C/C++

Выражения в языке C/C++ – это некоторая последовательность операндов и операций. Операндами могут выступать переменные, константы или другие выражения. Выражение может состоять из одной или более операций и определять целый ряд элементарных шагов по преобразованию данных. Компилятор соблюдает строгий порядок интерпретации выражений, определяемый приоритетом операций и правилами ассоциативности. Этот порядок может быть изменен, если отдельные части выражения взять в круглые скобки. Элементарная операция по преобразованию данного задается знаком операции.

По числу операндов, участвующих в операции, различают унарные (один операнд), бинарные (два операнда) и тернарные (три операнда) операции. По типу выполняемой операции различают арифметические, поразрядные логические, присваивания, отношения и другие.

11.3.1. Арифметические операции

В язык C/C++ включен стандартный набор арифметических операций, каждая из которых задается своим знаком:

– сложение (+);
– вычитание (-);
– умножение (*);
– деление (/);
– получение остатка от деления (%);
– инкремент (++);
– декремент (--).

Операции инкремента (декремента) увеличивают (уменьшают) операнд на 1 и могут применяться только к переменным. Существует две формы записи этих операций: префиксная `++<операнд>` (`--<операнд>`)

и постфиксная <операнд>++ (<операнд>--). В префиксной форме значение операнда сначала увеличивается (уменьшается) на 1 и увеличенное (уменьшенное) значение участвует в выражении. В постфиксной форме сначала используется значение в выражении, после чего значение увеличивается (уменьшается) на 1. Например:

```
// Пример использования префиксной и постфиксной формы инкремента

int main(){
     int a = 5;
     int b = 5;
     int c = ++a - b; //в результате c = 1, a = 6
     int d = a++ - b; // в результате d = 1, a = 7
     return 0;
}
```

Префиксная или постфиксная форма записи инкремента (декремента) играет роль в составных выражениях. В выражениях `++a`, `--a`, `a++`, `a--` различий между префиксной и постфиксной формы нет.

11.3.2. Операции присваивания

Операция простого присваивания «=» выполняется над двумя операндами, т. е. является бинарной, и несет в себе следующий смысл: в ячейку с адресом, ассоциированным с идентификатором, указанным слева от оператора «=», скопировать данное, указанное справа от оператора «=». Особенностью языка C/C++ является то, что кроме пересылки значения правого операнда в левый, операция присваивания дополнительно имеет еще и значение результата выполнения, совпадающее с присваиваемым значением. В языке C/C++ оператор присваивания правоассоциативен, т. е. вычисляется справа налево. Например:

```
signed int var_a = 1, var_b = 2, var_c = 3, var_d = 4;
var_a = var_b = var_c = var_d;
```

Выражение `var_a=var_b=var_c=var_d` будет выполняться в следующей последовательности: в переменную `var_c` скопировать значение переменной `var_d`, в переменную `var_b` скопировать значение переменной `var_c`, в переменную `var_a` скопировать значение переменной `var_b`. Таким образом, все переменные станут равны `var_d`. Для изменения порядка выполнения операции присваивания используют круглые скобочки `( )`. Так, в выражении

```
((var_a= var_b)= var_c)= var_d
```

переменная var_a принимает значение var_b, потом var_c, потом var_d.

```
signed int var_a = 1, var_b = 2, var_c = 3, var_d = 4;
((var_a = var_b) = var_c) = var_d;
```

В этом примере операции выполняются в порядке: var_a = var_b, var_a = var_c, var_a = var_d. Переменные var_b и var_c останутся неизменными, в то время как var_a станет равно var_d.

Так же язык C/C++ поддерживает операции совмещенного присваивания (или комбинированного присваивания): «+=», «-=», «*=», «/=», «%=», «<<=», «>>=», «&=», «|=», «=». Операции совмещенного присваивания имеют следующий синтаксис:

```
операнд_1 операция= операнд_2
```

Что означает: в операнд_1 скопировать результат указанной операции над операнд_1 и операнд_2.

Так, в результате операции присваивания со сложением var_a += var_b переменная var_a примет значение суммы var_a и var_b, что эквивалентно выражению var_a = var_a + var_b.

Приведем пример использования операций совмещенного присваивания:

```
signed short int a = 1;
     a += 10; // a = 1+10 = 11
     a -= 5;  // a = 11-5 = 6
     a *= 3;  // a = 6 * 3 = 18
     a /= 2;  // a = 18 / 2 = 9
     a %= 5;  // a = 9 % 5 = 4
     a >>= 2; // a = 4 >> 2 = 1
     a <<= 4; // a = 1 << 4 = 16
     a |= 3;  // a = 16 | 3 = 19
     a &= 6;  // a = 19 & 6 = 2
     a ^= 14; // a = 2 ^ 14 = 12
```

11.3.3. Поразрядные операции

Поразрядные операции выполняются над отдельными битами двоичного представления данного, а так же осуществляют поразрядный сдвиг всего двоичного значения вправо или влево. Поразрядные операции применимы только к целым данным, результатом их выполнения так же является целое.

Поразрядные логические операции

Язык C/C++ поддерживает следующие поразрядные логические операции:

1. «&» – поразрядная конъюнкция;
2. «|» – поразрядная дизъюнкция;
3. «^» – поразрядное сложение по модулю 2;
4. «~» – поразрядная инверсия.

Результаты побитовых логических операций в зависимости от значений операндов (таблица истинности) приведены в табл. 13. нверсия является унарной операцией, т. е. требует только одного операнда. Результат образуется поразрядной инверсией всех битов операнда.

В качестве примера приведем фрагмент программного кода и табл. 14, поясняющую выполняемые поразрядные операции:

```
signed char var_a = 13;
signed char var_b = - 12;
signed char var_c = var_a & var_b; // в результате c = 4
signed char var_d = var_a | var_b; // в результате d = - 3
var_d = ~ var_d;                   // в результате d = 2
```

Операции сдвига

Язык C/C++ включает две операции поразрядного линейного сдвига:

«<<» – сдвиг влево операнда, стоящего слева от знака сдвига, на количество разрядов, указанное справа от знака сдвига. Освобожденные справа биты заполняются нулями (рис. 12).

«>>» – сдвиг вправо операнда, стоящего слева от знака сдвига, на количество разрядов, указанное справа от знака сдвига. Метод заполнения освобождающихся левых битов зависит от того, какой тип результата получен после преобразования первого операнда. Если тип `unsigned`, то свободные левые биты заполняются нулями. В противном случае они заполняются копией знакового бита (рис. 13).

Оба операнда в операциях сдвига должны быть целыми. Если операнд справа от знака сдвига (количество бит, на которое должно быть сдвинуто двоичное число) отрицателем, результат операции сдвига не определен.

Операция сдвига влево равносильна операции умножения на число 2^n, а операция сдвига вправо равносильна операции деления на число 2^n, где n – число единичных сдвигов. Однако операции сдвига выполняются намного быстрее, чем умножение и деление. Приведем пример умножения и деления числа на 4 с использованием операций сдвига (табл. 15).

Таблица 13

Таблица истинности поразрядных логических операций

Бинарные поразрядные операции					Операция инверсии	
Операнд 1	Операнд 2	&	|	^	Операнд	~
0	0	0	0	0	0	1
0	1	0	1	1	1	0
1	0	0	1	1		
1	1	1	1	0		

Таблица 14

Пояснение к примеру о поразрядных операциях

№ разряда / Выражение	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	Комментарий
`signed char var_a = 13;`	0	0	0	0	1	1	0	1	Число 13
`signed char var_b = - 12;`	1	1	1	1	0	1	0	0	Число –12 в дополнительном коде
`signed char var_c=var_a & var_b;`	0	0	0	0	0	1	0	0	Число 4, получено конъюнкцией соответствующих разрядов чисел 13 и –12
`signed char var_d=var_a | var_b;`	1	1	1	1	1	1	0	1	Число -3 в дополнительном коде, получено дизъюнкцией соответствующих разрядов чисел 13 и –12
`var_d = ~ var_d;`	0	0	0	0	0	0	1	0	Число 2, получено инверсией числа –3

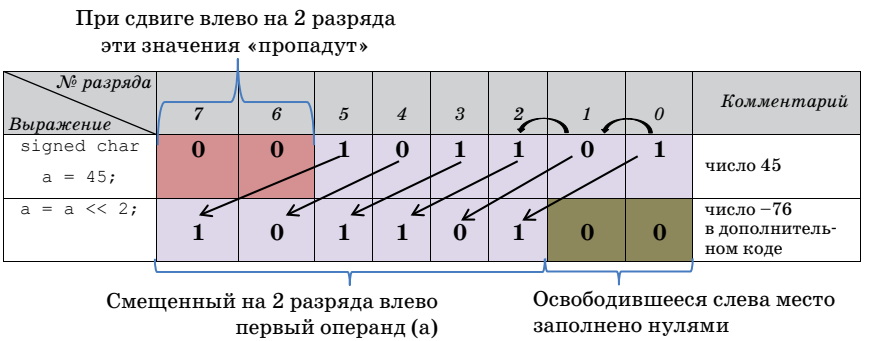


Рис. 12. Поразрядный сдвиг влево

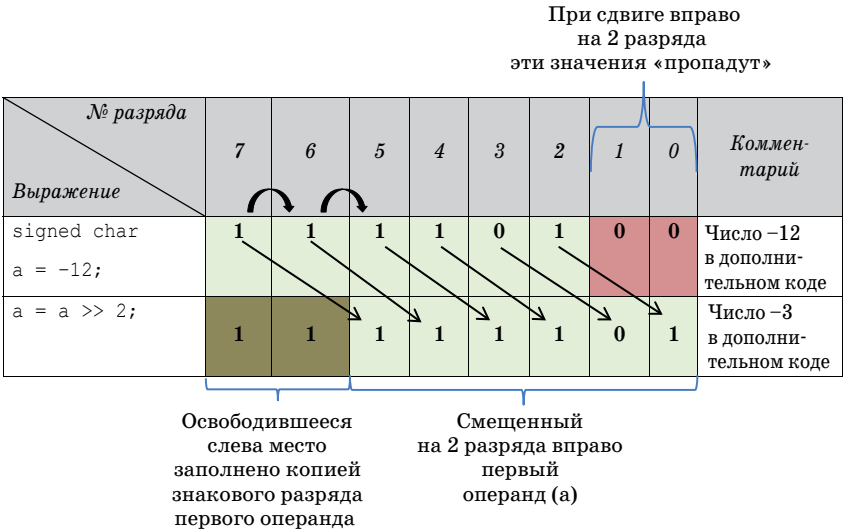


Рис. 13. Поразрядный сдвиг вправо

Таблица 15

Пример поразрядного сдвига в C/C++

№ разряда / Выражение	7	6	5	4	3	2	1	0	Комментарий
`signed char a= 16;`	0	0	0	1	0	0	0	0	число 16
`a = a >> 2;`	0	0	0	0	0	1	0	0	число 4, 16 / (2^2) = 4
`signed char b = 3;`	0	0	0	0	0	0	1	1	число 3
`b = b << 2;`	0	0	0	0	1	1	0	0	число 12, 3 * (2^2) = 12

Циклический сдвиг

Циклическим называют побитовый сдвиг, при котором освободившиеся разряды заполняются значениями, «вытесненными» из разрядной сетки числа. Язык C/C++ не предоставляет операторов циклического сдвига, однако его можно реализовать, используя операторы линейного сдвига и поразрядные логические операции. Приведем пример возможной реализации левого циклического сдвига в языке C/C++:

```
// Пример возможной реализации циклического сдвига ВЛЕВО
signed char rol(const signed char a, const unsigned char n)
{
// Функция возвращает результат циклического сдвига
// числа a на n разрядов ВЛЕВО

return (a<<n)|(static_cast <unsigned char> (a)>>(sizeof(a)*8-n));

}
```

В приведенной функции циклический сдвиг влево реализуется комбинацией поразрядных операций. Значение `a` сначала сдвигается на `n` разрядов влево, при этом старшие биты значения теряются. Для того, чтобы сохранить потерянные значения старших битов, значение `a` сдвигается вправо на количество разрядов, равное разности размера в битах, занимаемого данным, и `n`, что реализовано выражением `(sizeof(a) * 8 - n)`. По правилам сдвига вправо целого числа со знаком, освободившиеся слева значения заполняются путем размножения знака, т. е. в случае отрицательного числа – единицей, что приведет к дальнейшим ошибкам программы. Поэтому здесь необходимо преобразовать число к беззнаковой форме представления. Значение циклического сдвига влево получается «объединением» полученных при сдвигах значений с помощью поразрядной дизъюнкции. Поясним особенности работы функции для значений `a = 107`, `n = 3` (табл. 16).

Таблица 16

Пояснение к программе циклического сдвига влево

№ разряда / Выражение	7	6	5	4	3	2	1	0	Комментарий
`const signed char a = 107;`	0(z)	1	1	0	1	0	1	1	число +107
`a << n`	0(z)	1	0	1	1	0	0	0	число +88
`static_cast <unsigned char> (a)`	0	1	1	0	1	0	1	1	число 107
`static_cast <unsigned char> (a) >> (sizeof(a) * 8 - n)`	0	0	0	0	0	0	1	1	число 3
`(a< <n)|(static_ cast <unsigned char> (a)>> (sizeof(a)*8-n))`	0(z)	1	0	1	1	0	1	1	число +91

Пример возможной реализации правого циклического сдвига в языке C/C++ приведен ниже:

```
// Пример возможной реализации циклического сдвига ВПРАВО
signed char ror(const signed char a, const unsigned char n)
{
// Функция возвращает результат циклического сдвига
// числа a на n разрядов ВПРАВО

return (static_cast <unsigned char> (a)>>n)|(a<<(sizeof(a)*8-n));
}
```

Для реализации циклического сдвига вправо значение `a` преобразуется в беззнаковую форму и сдвигается на `n` разрядов вправо, при этом младшие биты значения теряются. Для того, чтобы сохранить потерянные значения младших битов, значение `a` сдвигается влево на количество разрядов, равное разности размера в битах, занимаемого данным, и `n`. Значение циклического сдвига вправо получается «объединением» полученных при сдвигах значений с помощью поразрядной дизъюнкции. Для `a = 107`, `n = 3` особенности работы функции представлены в табл. 17.

11.3.4. Логические операции и операции сравнения

Логические операции и операции сравнения используются в логических выражениях и как результат возвращают данное типа `bool`: 0 (`false`) либо 1 (`true`). Язык C/C++ поддерживает три логических операции и 6 операций сравнения. Это бинарные операции

Таблица 17

Пояснение к функции циклического сдвига вправо

№ разряда / Выражение	7	6	5	4	3	2	1	0	Комментарий
`const signed char a = 107`	0(z)	1	1	0	1	0	1	1	Число +107
`static_cast <unsigned char> (a)`	0	1	1	0	1	0	1	1	Число 107
`static_cast <unsigned char> (a) >> n`	0	0	0	0	1	1	0	1	Число 13
`a << (sizeof(a) * 8 - n)`	0(z)	1	1	0	0	0	0	0	Число +96
`(static_cast <unsigned char> (a)>>n)| (a<<(sizeof(a)*8-n));`	0(z)	1	1	0	1	1	0	1	Число +109

логическое И (&&) и логическое ИЛИ (||), операции сравнения (>, >=, <,<=, ==,!=) и унарная операция логическое НЕ (!). Операндами логических операций и операций сравнения могут выступать целые данные, числа с плавающей точкой, а так же указатели. При выполнении бинарных логических операций вычисляется первый операнд и, если его значения достаточно для определения результата, то второй операнд вычисляться не будет.

Результатом операции логическое И (&&) будет 1, если оба операнда отличны от нуля, и 0, если хотя бы один из операндов равен нулю. Если значение первого операнда равно нулю, то значение второго операнда не вычисляется.

Логическое ИЛИ (||) возвращает 1 в случае, если хотя бы один из операндов не равен нулю. Если оба операнда равны нулю, то и результат логического ИЛИ (||) равен нулю. Если первый операнд не равен нулю, то значение второго операнда не вычисляется. Например: (5>2)||(3<1), так как результат вычисления выражения в первой скобке равен 1 (true), вне зависимости от результата выражения во второй скобке результат будет 1 (true).

Логическое отрицание, или логическое НЕ, требует одного операнда, и возвращает true, если операнд равен false, и false, если операнд равен true.

11.4. Преобразование типов в выражениях, неявное и явное приведение типов

11.4.1. Неявное приведение типов

Т.к. язык C/C++ не является жестко типизированным языком, операнды бинарных операций могут быть разного типа. В таком случае перед выполнением операции компилятор предварительно приводит операнды к одному типу. Такое преобразование называют неявное приведение типов. Преобразования выполняются в соответствии с правилами приведения типов [11]:

1. Операторы разных типов приводятся к «старшему» из операндов, т. е. более длинному типу. Ниже перечислены типы в порядке возрастания старшинства, начиная с самого младшего:

```
char=>short int=>int=>unsigned int=>long int=> unsigned long=>
float=>double=>long double;
```

2. Если в выражение входят типы char или short, в знаковой или беззнаковой форме, они автоматически приводятся к типу int

(или `unsigned int`). В некоторых версиях языка С тип `float` автоматически приводится к типу `double`.

3. При выполнении операции присваивания результат приводится к типу переменной слева от знака присваивания. В этом случае возможно преобразование старшего типа к младшему, что может привести к потере точности, а в некоторых случаях и к получению ошибочного результата.

4. При использовании в качестве аргументов функции типов `char` или `short`, они автоматически приводятся к типу `int`, а тип `float` – к `double`. Однако автоматическое повышение типов можно отменить, используя прототипирования функций.

Преобразование младшего целого типа к старшему целому типу требует расширения значения. В таком случае старшие биты могут заполняться нулями (так называемое «размножение нуля»), либо старшим битом преобразуемого значения (так называемое «размножение знака»). Преобразование беззнакового целого типа в старший целый тип выполняется размножением нуля, а знакового – размножением знака. Пример, расширения целого типа с размножением знака пояснен в табл. 18.

Преобразования целого числа со знаком в целое число без знака того же размера (например, `signed int` в `unsigned int`) и наоборот не меняют битовый шаблон числа, но данное интерпретируется по другому. При этом возможно появление ошибок. Например, в табл. 19 рассмотрено вычисление выражений с неявным приведением типов, которое в случае переменной `var_c` дает результат, который непредвиден программистом и может расцениваться как ошибочный, хотя в случае переменной `var_d` ответ корректен.

Таблице 18

Пример расширения целого типа с размножением знака

№ разряда / Выражение	31	30	29	28	...	10	9	8	7	6	5	4	3	2	1	0	Комментарий
`signed char var_a = 6;`									0(z)	0	0	0	0	1	1	0	Число +6
`signed int var_b = var_a;`	0(z)	0	0	0	...	0	0	0	0	0	0	0	0	1	1	0	Число +6
`signed char var_c = -6;`									1(z)	1	1	1	1	0	1	0	Число –6 в доп.коде
`signed int var_d = var_c;`	1(z)	1	1	1	...	1	1	1	1	1	1	1	1	0	1	0	Число –6 в доп.коде

Таблица 19

Пример выполнения операторов с преобразованием целых типов данных со знаком и без знака одинакового размера

№ разряда / Выражение	7	6	5	4	3	2	1	0	Комментарий
`unsigned char var_a = 18;`	0	0	0	1	0	0	1	0	Число 18
`signed char var_b = -20;`	1(z)	1	1	0	1	1	0	0	Число –20 в доп. коде
`unsigned char var_c = var_a + var_b;`	1	1	1	1	1	1	1	0	Число 254 (результат вычислений непредвиден)
`unsigned char var_d = var_a - var_b;`	0	0	1	0	0	1	1	0	Число 38

Таблица 20

Пример неявного понижения типа

№ разряда / Выражение	31	30	29	28	...	10	9	8	7	6	5	4	3	2	1	0	Комментарий
`signed int var_a = 150;`	0(z)	0	0	0	...	0	0	0	1	0	0	1	0	1	1	0	Число +150
`signed char var_b = var_a;`									1(z)	0	0	1	0	1	1	0	Число –106 в доп. коде

Преобразование старших целых типов в младшие выполняется отбрасыванием старших байтов, что так же может привести к появлению грубых ошибок. При этом корректный ответ получается в том случае, если значение преобразуемого целого помещается в разрядную сетку младшего типа. Пример некорректного результата понижения типа приведен в табл. 20.

Преобразование данных (целых или с плавающей точкой) к типам с плавающей точкой производится по специальным библиотечным функциям.

11.4.2. Явное приведение типов

Язык C/C++ так же дает программисту возможность явно указать компилятору, в какой тип необходимо преобразовать данное в выражении. Явное приведение типов используют, например, в тех

случаях, когда по правилам автоматического приведения типов будет потеряна точность вычислений. Например

```
signed int var_a = 5;
signed int var_b = 10;
double var_c = var_a / var_b;            // ответ 0
double var_d = (double) var_a / var_b;   //ответ 0.5
```

В приведенном примере в результате выполнения выражения `var_c = var_a / var_b` переменная `var_c` примет значение `0`, т.к. справа от знака «=» выполняется целочисленное деление без приведения типов, результат которого – целочисленный `0` – приводится к типу `double`. При явном указании приведения типов `var_d = (double) var_a / var_b` выполняется явное преобразование переменной целого типа `var_a` к `double`, и автоматическое преобразование `var_b` к старшему в выражении типу, т. е. `double`. Вычисления выполняются в формате с плавающей точкой и полученный результат `0.5` копируется без преобразований в переменную `var_d`.

В языке С синтаксис явного преобразования типов следующий:

```
(тип) выражение
```

Результатом операции `(тип) выражение` является значение `выражения`, преобразованное в указанный `тип`. Приведенную операцию называют преобразованием в стиле С.

В языке С++ операции явного приведения типов выполняются с использованием операторов: `static_cast`, `reinterpret_cast`, `const_cast`, `dynamic_cast`. Для преобразований простейших типов данных, рассматриваемых в пособии, используется оператор явного преобразования `static_cast`, имеющий следующий синтаксис:

```
static_cast <тип> (выражение)
```

После оператора `static_cast` в угловых скобках указывается `тип`, к которому необходимо привести `выражение`, указанное в круглых скобках. Пример, приведенного выше, но написанный в стиле С++, будет выглядеть как:

```
signed int var_a = 5;
signed int var_b = 10;
double var_c = var_a / var_b;                        // ответ 0
double var_d = static_cast<double> (var_a)/var_b; //ответ 0.5
double var_f = static_cast<double> (var_a/var_b); //ответ 0
```

В последнем выражении в примере, `var_f = static_cast<double> (var_a / var_b)`, сначала выполняется выражение, указанное в скобках и представляющее целочисленное деление, только после этого результат преобразуется в `double`.

После введения операций явного преобразования в языке C++, преобразования в стиле C стали нежелательными. Одна из причин введения такого ограничения состоит в том, что при использовании оператора `static_cast` компилятор следит не только за возможностью приведения типа, но так же за безопасностью этой операции, что повышает надежность программы. При использовании объектно-ориентированного программирования, преобразование в стиле C может привести к трудноуловимым ошибкам, а преобразования в стиле C++ легко найти во всём проекте (проектах) и выявить потенциальные места ошибок.

11.5. Приоритет операций и порядок их выполнения

Операции в выражениях выполняются в строго определенном порядке, определяемом приоритетом операций. Так же, как и в математике, операция умножения в языке C/C++ имеет более высокий приоритет, чем операция сложения, следовательно и выполняется раньше. Если операции в выражении имеют одинаковый приоритет, они выполняются в порядке их следования. При этом для большинства операций определен порядок следования слева-направо (а операция присваивания «=» является исключением и выполняется справа-налево). Так, порядок вычисления в операторе:

```
var_y = 50-30/5*2
```

будет следующий:

1. `30/5` — у умножения и деления в C/C++ одинаковый приоритет, и он выше, чем у вычитания. Следовательно, первой будет выполнено деление, т.к. в выражении оно указано левее, чем умножение, результат операции равен 6;
2. `6*2` — результат первой операции переходит во вторую по приоритету операцию – умножение, результат 12;
3. `50 - 12` — далее вычисляется разность, результат 38;
4. `var_y = 38` — операция присваивания выполняется справа налево и имеет наименьший приоритет.

Изменить порядок выполнения операций можно с помощью оператора круглые скобки «`( )`». В выражении `var_y = (50-30)/(5*2)` сначала будут выполнены вычитание `(50-30)` и произведение `(5*2)`, и результаты этих операций являются операндами деления. Результат вычисления `2.` будет присвоен в переменную `var_y`.

Последовательность выполнения операций в выражениях называют ассоциативностью. Правила ассоциативности применяются,

когда на один и тот же операнд «претендуют» две операции. Например, в выражении `30/5*2`, операнд `5` может использоваться как в операции «*», так и в операции «/», которые имеют одинаковый приоритет. В зависимости от порядка выполнения операций, результат может быть равен или 12, или 3. Но для операций умножения и деления строго определено правило ассоциативности слева-направо, поэтому сначала будет выполнено деление, а после – умножение. В выражении `(50-30)/(5*2)` две операции, заключенные в скобки и имеющие одинаковый приоритет, не претендуют на один и тот же операнд, поэтому правила ассоциативности не применяются, а порядок их выполнения зависит от реализации системы.

Приоритет основных операций в языке C/C++, а так же их ассоциативность, приведены в табл. 21. В таблице операции приведены в порядке уменьшения приоритета, т. е. меньшее значение уровня соответствует более высокому приоритету. Внутри одного уровня операции имеют одинаковый приоритет. Если не указано иное, предполагается, что операции бинарны, т. е. в них участвуют два операнда.

Таблица 21

Приоритет основных операций в языке C/C++

Уровень	Операции	Ассоциативность
1	: : (изменение области видимости)	Слева-направо →
2	`( )` (вызов функции) `[]` (взятие элемента массива) . (выбор элемента по ссылке) -> (выбор элемента по указателю) ++ (постфиксный инкремент) -- (постфиксный декремент)	Слева-направо →
3	Унарные операции: + (унарный плюс) - (унарный минус) ~ (побитовое отрицание) ! (логическое отрицание) ++ (префиксный инкремент) -- (префиксный декремент) `sizeof()` *(разыменование) & (взятие адреса) операции приведение типа	Справа-налево ←
4	* (умножение) / (деление) % (получение остатка от деления)	Слева-направо →

Окончание табл. 21

Уровень	Операции	Ассоциативность
5	+ (сложение) - (вычитание)	Слева-направо →
6	>> (побитовый сдвиг вправо) << (побитовый сдвиг влево)	Слева-направо →
7	< (меньше) > (больше) <= (меньше или равно) >= (больше или равно)	Слева-направо →
8	== (равно) != (неравно)	Слева-направо →
9	& (поразрядная конъюнкция, И)	Слева-направо →
10	^ (поразрядная сумма по модулю 2, XOR)	Слева-направо →
11	| (поразрядная дизъюнкция, ИЛИ)	Слева-направо →
12	&& (логическое И)	Слева-направо →
13	|| (логическое ИЛИ)	Слева-направо →
14	?: (условный оператор, тернарный)	Справа-налево ←
15	= (прямое присваивание) Сложное присваивание:+= -= *= /= %= >>= <<= &= ^= |=	Справа-налево ←
16	, (последовательное вычисление)	Слева-направо →

12. МАССИВЫ И СТРУКТУРЫ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С/С++

12.1. Массивы

Из простейших типов данных строятся более сложные, составные данные, среди которых следует выделить массивы и структуры.

Массив – совокупность данных одного типа, расположенных в смежных ячейках памяти, обращение к которым происходит по индексу. Различают одномерные, двумерные, n-мерные массивы. Для массива определены те же операции, что для его элементов. Объявление массива в языке С/С++ выполняется в соответствии с синтаксисом:

```
тип_данных  имя_массива[количество_элементов_массива];
```

`тип_данных` определяет тип всех элементов массива;

`имя_массива` – идентификатор, который в свою очередь представляет адрес в памяти начала области, занимаемой элементами массива.

`количество_элементов_массива` – константа (для статических массивов), определяющая количество элементов в массиве (размер массива, длина массива). Так как компилятор должен знать, сколько памяти выделить для хранения статического массива, при объявлении массива нужно указать постоянное значение `количество элементов_массива`, которое может быть представлено именованной или неименованной константой.

При объявлении массива согласно приведенному синтаксису в памяти компьютера выделяется объем, равный произведению `количество_элементов_массива` на количество байт, выделяемых для хранения данного типа `тип_данных`. Адрес первого бита из выделенной памяти связывается с `имя_массива`. Например:

```
const unsigned int ARRAY_SIZE = 5;    //константа- размер массива
signed int arr[ARRAY_SIZE];     // объявление массива
```

Нумерация элементов массива в языке С/С++ начинается с нуля. Для обращения к элементу массива необходимо указать имя массива, после которого в квадратных скобках – номер элемента массива:

```
arr[4] = 5;
```

В примере выполняется обращение к 4-му элементу массива. Так как нумерация элементов массива в языке С/С++ начинается с 0, то по сути это пятый по порядку элемент массива.

При объявлении массива без инициализации значений его переменных в ячейках памяти, выделенных под массив, лежат случай-

ные значения, которые, в то же время, могут быть использованы в операциях, что влечет за собой ошибки и непредвиденное поведение программы.

Для заполнения (инициализации) массива возможны несколько подходов.

1. Инициализация каждого элемента массива значением:

```
const unsigned int ARRAY_SIZE = 5;    //константа - размер массива
signed int arr[ARRAY_SIZE];     // объявление массива
arr[0] = 1;    arr[1] = 10; arr[2] = 100; arr[3] = 1000;
arr[4] = 10000;
```

2. Объявление массива с инициализацией, при этом необязательным является указание размера массива, а память выделяется в соответствии с количеством значений, указанных в фигурных скобках после знака «=»:

```
signed int arr[] = {1, 10, 100, 1000, 10000};
```

3. Заполнение массива в цикле:

```
const unsigned int ARRAY_SIZE = 5;    //константа - размер массива
signed int arr[ARRAY_SIZE];            // объявление массива
for (int i = 0; i < ARRAY_SIZE; i++) arr[i] = 3*i+1;
```

Элементами массива могут быть не только данные простейшего типа, но и другие массивы. Например:

```
const unsigned int ROW_SIZE = 2;
const unsigned int COL_SIZE = 3;
signed int arr[ROW_SIZE][COL_SIZE];
```

В приведенном примере объявлен массив `arr` из 2 элементов (`ROW_SIZE = 2`), каждый элемент массива в свою очередь представляет массив с 3 элементами (`COL_SIZE = 3`). Инициализация двумерных массивов возможна так же, как и для одномерных:

1. Инициализация каждого элемента массива значением:

```
const unsigned int ROW_SIZE = 2;
const unsigned int COL_SIZE = 3;
signed int arr[ROW_SIZE][COL_SIZE];
arr[0][0]= 1; arr[0][1] = 10; arr[0][2] = 100;
arr[1][0]= 2; arr[1][1] = 20; arr[1][2] = 200;
```

2. Объявление массива с инициализацией:

```
signed int arr[] = {{1, 10, 100}, {2, 20, 200}};
```

3. Заполнение массива в цикле:

```
const unsigned int ROW_SIZE = 2;
const unsigned int COL_SIZE = 3;
signed int arr[ROW_SIZE][COL_SIZE];
for (int i = 0; i < ROW_SIZE; i++)
```

```
    for (int j = 0; j < COL_SIZE; j++)
            arr[i][j] = i*j;
```

Особым видом массивов в языке C/C++ являются строки. Строка представляет собой массив с элементами типа `char`. Конец строки определяется нуль-символом `'\0'` и с этим связаны особенности обработки строк. Значение строки записывают в двойных кавычках, при этом компилятор автоматически добавляет нуль-символ в конце строки, а указание размера строки необязательно:

```
char a[] = "Hello, world!";
```

Приведем пример фрагмента программы для расчета количества элементов в стоке:

```
// объявление строки с инициализацией
char str_a[] = "Hello, world!";
// переменная len будет содержать длину строки
int len = 0;
// в цикле перебираются все элементы строки,
// пока не будет достигнут нуль-символ
while (a[len] != '\0') len++;
```

Следует отличать инициализацию строк и символов. Для инициализации символов используют одинарные кавычки. Для строк – двойные кавычки, указывающих компилятору о необходимости добавления нуль-символа в конце. Например объявление одиночного символа:

```
char var_a = '5';
```

Объявление строки символов из 2 элементов, `'5'` и `'\0'`:

```
char str_b[] = "5";
```

Так как строки представляют собой массивы символов, к каждому элементу строки можно обратиться по индексу:

```
char str_a = "Hello, world!";       // объявление строки символов
char var_a = str_a[7];              // обращение 4 символу строки
```

12.2. Структуры (записи)

Структуры – это составной тип данных, элементом которого является совокупность данных разного типа. Структура состоит из полей, каждое поле представляет собой данное какого либо типа, но разные поля структуры могут быть различного типа. Данные, соответствующие полям структуры, хранятся в смежных ячейках памяти. Одно из полей структуры является ключевым. Например, структура «Студент» может иметь поле «порядковый номер» целочисленного типа, поле «ФИО» строкового типа, поле «Оценка»

целого беззнакового типа. В структуре «Студент» ключевым полем может выступать поле «ФИО». Для определения структуры в языке C/C++ используется следующий синтаксис:

```
struct имя_структуры {
            описание структуры
    };
```

При определении структуры в языке C/C++ необходимо следить, чтобы оно заканчивалось точкой с запятой. Например:

```
struct Student {
    unsigned int ID;              // Порядковый номер студента
    char name[100];               // ФИО студента
    unsigned int grade;           // Оценка
};
```

Объявление элемента типа структуры соответствует объявлению переменной. Инициализация полей структуры возможна через оператор фигурные скобки, в которых значения следуют в соответствии с указанием полей в определении структуры. Для обращения к полям структуры указывается элемент структуры и через точку необходимое поле:

```
Student Ivanov = {1,"Иванов Иван Иванович" , 5};
printf("Порядковый номер студента: %d\n", Ivanov.ID);
printf("ФИО студента: %s\n", Ivanov.name);
printf("Оценка студента: %d\n", Ivanov.grade);
```

Структура может является и элементом массива. «Экзаменационная ведомость» является примером массива структур «Студент», т.к. представляет собой совокупность из некоторого количества элементов одного типа – «Студент».

```
Student examination_sheet[25];
examination_sheet[9] = { 9,"Иванов Иван Иванович" , 5 };
printf("Имя студента: %s\n", examination_sheet[9].name);
```

В процессе работы системы возникает необходимость создания новых типов данных, состоящих из простейших – данных, определяемых пользователем. Для них пользователь определяет диапазон значений и набор допустимых операций. Такие типы данных принято называть классы, а операции над ними – методами. Классы являются предметом объектно-ориентированного программирования и в данном пособии не рассматривается.

13. ИМЕНОВАНИЕ ДАННЫХ В ПРОГРАММНЫХ ПРОЕКТАХ

На сегодняшний день для разработки программ привлекаются команды программистов, в процессе написания и дальнейшего использования, программы многократно претерпевают изменения, дополнения, оптимизацию, исправление ошибок. Так как работу над одним и тем же программным кодом в процессе его жизненного цикла выполняют зачастую различные люди, крайне актуальным становиться вопрос разработки понятного и легко-читаемого кода. Одним из аспектов, позволяющих повысить читаемость программного кода, является осмысленное именование переменных [13].

Синтаксические правила именования переменных могут отличаться в зависимости от языка. В таких языках, как C/C++, MATLAB, имена переменных могут состоять из букв латинского алфавита, цифр и знака нижнего подчеркивания, но начинаться должны с буквы. Кроме ограничений, накладываемых на имена переменных синтаксисом языка, существуют приемы, позволяющие сделать программный код более читаемым.

Сравним два фрагмента программного кода:

```
aaa = sqrt(q * q - 4 * w*e);     discriminant = sqrt(b * b - 4 * a *c);
xxx[0] = (-q + aaa) / 2 / w;     root[0] = (-b + discriminant) / (2 * a);
xxx[1] = (-q - aaa) / 2 / w;     root[1] = (-b - discriminant) / (2 * a);
```

Оба фрагмента работают одинаково, однако второй из предложенных вариантов более понятен читателю, т.к. имена переменных сами, без дополнительного комментирования, поясняют программный код.

При именовании переменных принято соблюдать следующие рекомендации:

1. Имя переменной должно максимально точно отображать смысл, который эта переменная несет в программе. Например, переменная, хранящая текущую дату, может называться `currentDate`, имя студента – `studentName`.

2. Не следует использовать транслит при именовании переменных, лучше подобрать необходимое слово на английском языке.

3. Короткие имена лучше использовать для локальных переменных, используемых в небольшом фрагменте кода.

4. Имена должны быть отформатированы так, чтобы их было легко читать. При этом программистами используются различные нотации. Верблюжья нотация (CamelCase) – все слова в имени

пишутся слитно, а разделение между словами выполняется за счет разницы регистров:

```
unsigned int numberOfStudents;
```

Змеиная нотация (snake_case) – все слова в имени пишутся строчными буквами, а разделение слов осуществляется знаком нижнего подчеркивания «_». Например:

```
unsigned int number_of_students;
```

5. Возможно использование общепринятых суффиксов и семантических префиксов. Для переменных, содержащих, например, максимальное или минимальное значение, сумму и т.п., в конце имени можно добавить суффикс типа `Total`, `Sum`, `Count`, `Average`, `Max`, `Min`, `Record`, `String` или `Pointer`, так – же называемый *Спецификатором вычисляемых значений*. Общепринятыми являются префиксы `i` – индекс массива, `p` – указатель, `max/min` – индекс последнего/первого элемента массива или другого списка, `first/last` – элемент массива, обрабатываемый первым/последним, `c` – количество (счетчик, counter) `g` – глобальная переменная. Например

```
unsigned int studentCount;  // счетчик количества студентов
double functionMax;         // экстремум (максимум) функции
int *pa;                    //указатель на переменную a
```

6. Для переменных логического типа зачастую удобно использовать имена, так же как и данные этого типа, подразумевающие «Истину» или «Ложь». Например,

```
bool  found;
bool  equal;.
bool  done;.
```

7. Константы принято именовать прописными буквами, например,

```
const double PI = 3.14159265;
```

Следует избегать использовать в имени переменных символов, которые можно спутать с другими символами: (`1` и `I`), (`.` и `,`), (`0` и `O`), (`2` и `Z`), (`;` и `:`), (`S` и `5`), (`G` и `6`).

8. При объявлении переменной в комментариях можно описать аспекты переменной, которые невозможно выразить в ее имени, например, единицы измерения, диапазоны значений, ограничения.

ЗАКЛЮЧЕНИЕ

Типы и структура данных являются основополагающим понятием любых технологий и языков программирования. В зависимости от назначения, трактовка этих понятий в различных языковых средах может быть различной. Для построения эффективных систем обработки информации важно всегда учитывать реальные процессы, происходящие в вычислительной системе. Материал, приведенный в настоящем учебном пособии, носит достаточно общий характер и рассмотренные принципы организации данных применимы и используются практически во всех языках программирования, хотя каждый из них, естественно, имеет свои особенности.

Список использованной литературы

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

2. Шеннон К. Э. Математическая теория связи //Работы по теории информации и кибернетике / Пер. С. Карпова. – М.:ИИЛ, 1963. –830 с.

3. Степанов А. Н. Курс информатики для студентов информационно-математических специальностей. – СПб.: Питер, 2017. – 1088 с.

4. Ключарев А. А., Матьяш В. А., Щекин С. В. Структуры и алгоритмы обработки данных: учебн. пособие. – СПб.: ГУАП, 2003. – 172 с.

5. Лысиков Б. Г. Арифметические и логические основы цифровых автоматов. – Мн.: Высшая школа, 1980. – 336 с.

6. Довгий П.С., Поляков В.И. Арифметические основы ЭВМ: учеб.-метод. пособие. – СПб.: СПбГУ ИТМО, 2010. – 56 с.

7. ГОСТ Р 34.303-92. Наборы 8-битных кодированных символов. 8-битный код обмена и обработки информации.

8. Хабибулин И. Ш. Программирование на языке высокого уровня C/C++. – СПб.: БХВ-Петербург, 2006. – 512 с.

9. Международный стандарт ISO/IEC 9899:201x. Programming languages – C.

10. Международный стандарт ISO/IEC 14882:2014(E) Programming Language C++

11. URL: https://docs.microsoft.com/ (дата обращения: 23.11.2020).

12. Полубенцева М. И. C/C++. Процедурное программирование. – СПб.: БХВ-Петербург, 2017. – 432 с.

13. Стив Макконнелл. Совершенный код. Практическое руководство по разработке программного обеспечения / Пер. с англ. – М.: Русская редакция, 2019. – 896 с.

СОДЕРЖАНИЕ

Введение 3

1. Информация и данные в информационных процессах ... 4

2. Структура и принцип действия вычислительной машины. Принципы фон Неймана 9

3. Представление данных в компьютерных системах 13
   3.1. Уровни представления данных в компьютерных системах 13
   3.2. Простейшие (примитивные) типы данных 15
   3.3. Основные составные структуры данных – массивы и записи 17

4. Числовые данные 18
   4.1. Основные виды чисел 18
   4.2. Позиционная система счисления 19
   4.3. Перевод чисел из одной системы счисления в другую 22

5. Формы представления чисел в разрядной сетке вычислительной машины 27
   5.1. Разрядная сетка. Представление целых чисел без знака 27
   5.2. Представление целых чисел со знаком 28
   5.3. Числа с фиксированной точкой 29
   5.4. Числа с плавающей точкой 30

6. Представление чисел со знаком в вычислительной машине 36
   6.1. Числа в прямом и дополнительном кодах 36
   6.2. Обработка переполнения разрядной сетки при сложении двоичных чисел 41

7. Арифметические операции с числами с плавающей точкой. Сложение чисел с плавающей точкой 45

8. Особенности выполнения умножения в вычислительной машине 47

9. Диапазон представления чисел в различных форматах для двоичной системы счисления 51

10. Представление символьных данных в компьютерных системах. Кодовые таблицы 54
   10.1. Основные принципы кодирования символов 54

10.2. Кодовая таблица ASCII 56
10.3. Национальные восьмибитовые кодовые таблицы на основе ASCII 59
10.4. Кодирование символов в Unicode (Юникод) 63
11. Особенности представления данных в языке программирования C/C++ 66
11.1. Простейшие типы данных и их внутреннее представление 66
11.2. Идентификаторы. Переменные. Константы 74
11.3. Выражения и операции в языке C/C++ 79
11.4. Преобразование типов в выражениях, неявное и явное приведение типов 87
11.5. Приоритет операций и порядок их выполнения ... 91
12. Массивы и структуры в языке программирования C/C++ 94
12.1. Массивы 94
12.2. Структуры (записи) 96
13. Именование данных в программных проектах 98
Заключение 100
Список использованной литературы 101

Учебное издание

Ключарёв Александр Анатольевич,
Фоменкова Анастасия Алексеевна

ТИПЫ И СТРУКТУРЫ ДАННЫХ В ИНФОРМАТИКЕ И ПРОГРАММИРОВАНИИ

Учебное пособие

ISBN: 978-5-8088-1561-2

Публикуется в авторской редакции
Компьютерная верстка *Н. Н. Караваевой*

Подписано в печать 25.03.21. Формат 60×84 1/16.
Усл. печ. л. 6,1. Уч.-изд. л. 6,5. Тираж 50 экз. Заказ № 61.

Редакционно-издательский центр ГУАП
190000, Санкт-Петербург, Б. Морская ул., 67