Содержание

ПРЕДИСЛОВИЕ

В 2012 г. языку Си исполняется 40 лет. Как латинский язык явился основой многих европейских языков, так и язык Си стал родоначальником языков Си++, Java, Perl, C#, PHP, JavaScript и т. д. В отличие от мертвой латыни, язык Си – не только живой язык, но и наиболее распространенный и эффективный из универсальных языков программирования. Программы на языке Си исполняются почти на всех компьютерах, он работает со средой программирования UNIX, и сама операционная система UNIX написана на нем. На Си написано множество библиотечных функций и утилит. Сотни тысяч программистов знают язык Си, он неотделим от общечеловеческой культуры программирования. Любой программист должен владеть языком Си, чтобы профессионально работать в области информационных технологий и понимать своих коллег. В учебных программах и стандартах высших и средних специальных учебных заведений для большинства естественно-научных специальностей предусмотрено изучение программирования на языке Си.

Язык программирования Си создан в 1972 г. сотрудником фирмы Bell Laboratories Деннисом Ритчи (Dennis M. Ritchie) при разработке операционной системы UNIX. Язык проектировался как инструмент для системного программирования с ориентацией на разработку хорошо структурированных программ. Удачное сочетание лаконичности конструкций и богатства выразительных возможностей позволило языку Си быстро распространиться и стать наиболее популярным языком прикладного и системного программирования. Компиляторы языка Си работают почти на всех типах современных ЭВМ в операционных системах Windows, UNIX-подобных ОС (FreeBSD, Linux), Solaris, Mac OS и др.

В отличие от многих предшествующих языков (Ада, Алгол-60, Алгол-68 и т. д.), которые вступали в силу после принятия соответствующих национальных и международных стандартов, язык Си вначале был создан как рабочий инструмент, не претендующий на широкое применение. Стандарта на язык Си до 1989 г. не сущест-

вовало, и в качестве формального описания разработчики компиляторов использовали первое издание книги Б. Кернигана и Д. Ритчи, вышедшее в США в 1978 г. (переведена на русский язык в 1985 г. [1]). Роль неформального стандарта языка Си сохранилась за этой книгой и в настоящее время. Не случайно в литературе и документации по компиляторам ссылка на эту работу обозначается специальным сокращением K&R.

Второе издание книги Б. Кернигана и Д. Ритчи [2] описывает язык Си в стандартизованном Американским институтом национальных стандартов виде (стандарт ANSI языка Си). В настоящее время, кроме стандарта ANSI C, разработаны международный стандарт ISO C (International Standard Organization C) и стандарт 1999 года **C99** – современный стандарт языка программирования Си. Определен в ISO/IEC 9899:1999, современная версия – ISO/IEC 9899:1999/ Cor 3:2007 от 2007-11-15. Эти версии стандарта близки друг к другу, и на различиях между стандартами нет необходимости останавливаться до возникновения разногласий в толковании той или иной конструкции языка либо при оценке стандартности конкретного компилятора. Эти ситуации выходят за рамки курса по программированию на языке Си. В случае необходимости получения справок по стандартам языка Си следует обращаться к специальным публикациям. Неформальное применение книги K&R в качестве стандарта до 1989 г. и последующая ее переработка авторами в соответствии с принятым стандартом ANSI привели к тому, что ее и сейчас можно рассматривать как достоверный источник при получении справок по языку Си.

Настоящий учебник предназначен для изучения программирования на стандартном языке Си. Ориентация сделана как на изложение синтаксиса и семантики конструкций языка, так и на их практическое использование при решении типовых задач программирования. После описания в главе 1 основных понятий языка Си рассмотрены средства представлений базовых конструкций структурного программирования, возможности которых в главе 2 иллюстрируются на простых вычислительных задачах. Глава 3 содержит подробное описание препроцессорных средств компилятора языка Си, которые активно используются при последующем изучении методов и приемов программирования на языке Си. Следующая глава и посвящена незаменимым в системном программировании понятиям – объектам и адресам (указателям). Аппарат указателей используется затем при обработке массивов и строк. Центральное

место занимает глава 5, посвященная функциям. Здесь возможности функций рассмотрены подробно и с нужной полнотой. Особое внимание уделено взаимосвязи функций с указателями, а также классам памяти, которые вводятся в связи с организацией многофайловых программ, включающих много функций. Глава 6 рассматривает структурированные данные (структуры и объединения). Особенности работы с файлами, а также средства ввода-вывода показаны на типовых задачах в главе 7. В главе 8 приводятся сведения о подготовке и выполнении программ в среде семейства операционных систем UNIX.

Приводимые в учебнике программы сопровождаются результатами, полученными на ЭВМ. Более подробно с практическими приемами программирования на языке Си читатель может познакомиться, обратившись к «Практикуму по программированию на Си» [10].

Целью настоящего учебника является изложение методики и принципов корректного, структурированного программирования на языке Си. Программы, иллюстрирующие конструкции и возможности языка, написаны максимально понятно для читателя. Авторы нигде не гнались за эффективностью кода в ущерб его структурированности и простоты. Возможности современных компиляторов языка Си таковы, что они позволяют генерировать весьма эффективный код по тексту хорошо структурированной программы без специальных ухищрений программиста, направленных на повышение быстродействия или незначительную экономию памяти.

Книга написана на основе дисциплин, которые авторы в течение ряда лет преподавали в МИЭМе на факультете прикладной математики, на факультете автоматики и вычислительной техники и факультете повышения квалификации инженеров. Материал курса соответствует учебной программе дисциплины «Алгоритмические языки и программирование». Изучение указанной дисциплины, в частности языка Си, служит основой для курсов по математическому обеспечению ЭВМ и сетей, по операционным системам, построению компиляторов и системному программированию.

Авторы надеются, что книга поможет ликвидировать разрыв между техническими руководствами по реализации языка Си и потребностями в методическом обеспечении учебного процесса. Для чтения книги достаточно знать основы информатики. Поэтому пособие можно использовать как в вузе, так и в курсах информатики школ, гимназий, лицеев и техникумов. Необходимым условием освоения

материала книги является выполнение приведенных в ней примеров на любой ЭВМ, снабженной транслятором с языка Си.

Повышению качества рукописи способствовали замечания рецензентов.

Любые конструктивные замечания и предложения по улучшению учебника авторы с благодарностью примут и учтут в дальнейшем. Нам можно писать по адресу издательства.

Глава 1
БАЗОВЫЕ ПОНЯТИЯ ЯЗЫКА

Начиная изучать новый для вас алгоритмический язык программирования, необходимо выяснить следующие вопросы:

1. Каков алфавит языка и как правильно записывать его лексемы[1]?
2. Какие типы данных приняты в языке и как они определяются (описываются)?
3. Какие операции над данными допустимы в языке, как строятся с их помощью выражения и как они выполняются?
4. Какова структура программы, в какой последовательности размещаются операторы, описание и определения?
5. Как выводить (представлять пользователю) результаты работы программы?
6. Как реализованы оператор присваивания, условные операторы и операторы перехода?
7. Как вводить исходные данные для программы?
8. Какие специальные конструкции для организации циклов есть в языке?
9. Каков аппарат подпрограмм (процедур) и (или) подпрограмм-функций?

Затем следует приступать к составлению программ, углубляя в ходе программирования знание языка. Изложение материала в данном пособии почти соответствует описанной схеме изучения алгоритмических языков. Введя основные средства языка Си, будем рассматривать конкретные программы, а затем, переходя к новым классам задач, введем все конструкции языка и те средства, которые не упоминаются в перечисленных выше вопросах.

---

[1] Лексема – единица текста программы, которая при компиляции воспринимается как единое целое и по смыслу не может быть разделена на более мелкие элементы.

В начале первой главы рассмотрим алфавит, идентификаторы, константы, типы данных и операции языка. Этот базовый материал необходим для всех следующих глав. Не освоив перечисленных понятий, невозможно начинать программирование.

Традиционно перед изложением синтаксиса языка программирования авторы пособий дают неформальное введение, где на примерах иллюстрируют основные принципы построения программ на предлагаемом языке. Однако язык Си невелик, и его лексические основы можно рассмотреть весьма подробно уже в самом начале изучения. Поэтому начнем с алфавита и лексем.

1.1. Алфавит, идентификаторы, служебные слова

Алфавит. В алфавит языка Си входят:

- прописные и строчные буквы латинского алфавита (A, B, ..., Z, a, b, ..., z);
- цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
- специальные знаки: " , { } | [ ] ( ) + – / % \ ; ' . : ? < = > _ ! & * # ~ ^;
- неизображаемые символы («обобщенные пробельные символы»), используемые для отделения лексем друг от друга (например, пробел, табуляция, переход на новую строку).

В комментариях, строках и символьных константах могут использоваться и другие литеры (например, русские буквы).

Комментарий формируется как последовательность знаков (символов), ограниченная слева знаками /*, а справа – знаками */. Например:

```
/* Это комментарий */
```

В стандартном языке Си комментарии запрещено вкладывать друг в друга, то есть запись:

```
/* текст-1 /* текст-2 */ текст-3 */
```

ошибочна – «текст-3» не считается комментарием.

В современных версиях языка Си (C89, C9x) можно использовать «комментарий в строке», начинающийся с двух символов «//» и продолжающийся до конца строки.

В языке Си шесть классов лексем: свободно выбираемые и используемые идентификаторы, служебные (ключевые) слова, константы, строки (строковые константы), операции (знаки операций), разделители (знаки пунктуации).

Идентификатор. Последовательность букв, цифр и символов подчеркивания «_», начинающаяся с буквы или символа подчеркивания, считается идентификатором языка Си. Примеры идентификаторов:

```
KOM_16,     size88,     _MIN,     TIME,     time
```

Прописные и строчные буквы различаются, то есть два последних идентификатора различны.

Идентификаторы могут иметь любую длину, но компилятор учитывает не более 31 символа от начала идентификатора. В некоторых компиляторах это ограничение еще более жесткое, и учитываются только первые 8 символов любого идентификатора. В этом случае идентификаторы NUMBER_OF_ROOM и NUMBER_OF_TEST в программе будут неразличимы.

Служебные (ключевые) слова. Идентификаторы, зарезервированные в языке, то есть такие, которые нельзя использовать в качестве свободно выбираемых программистом имен, называют служебными словами. Служебные слова определяют типы данных, классы памяти, квалификаторы типа, модификаторы, псевдопеременные и операторы. В стандарте языка определены следующие служебные слова:

auto	**break**	**case**	**char**	**const**	**continue**	**default**	**do**
double	**else**	**enum**	**extern**	**float**	**for**	**goto**	**if**
inline	**int**	**long**	**register**	**restrict**	**return**	**short**	**signed**
sizeof	**static**	**struct**	**switch**	**typedef**	**union**	**unsigned**	**void**
volatile	**while**	**_Bool**	**_Complex**	**_Imaginary**			

Служебные слова в конструкциях языка используются по-разному.

Для обозначения типов данных они служат спецификаторами и квалификаторами типов. Последние уточняют свойства обозначенных типов данных.

К *спецификаторам типов* относятся:

- **char** – символьный;
- **double** – вещественный двойной точности с плавающей точкой;

- **enum** – перечисляемый тип (перечисление) – определение целочисленных констант, для каждой из которых вводятся имя и значение;
- **float** – вещественный с плавающей точкой;
- **int** – целый;
- **long** – целый увеличенной длины (длинное целое);
- **long long** – целый тип длиной не менее 64 бит;
- **short** – целый уменьшенной длины (короткое целое);
- **struct** – структура (структурный тип);
- **signed** – знаковый, то есть целое со знаком (старший бит считается знаковым);
- **union** – объединение (объединяющий тип);
- **unsigned** – беззнаковый, то есть целое без знака (старший бит не считается знаковым);
- **void** – отсутствие значения;
- **typedef** – вводит синоним обозначения типа (определяет сокращенное наименование для обозначения типа).

Квалификаторы типа:

- **const** – квалификатор объекта, имеющего постоянное значение, то есть доступного только для чтения;
- **volatile** – квалификатор объекта, значение которого может измениться без явных указаний программиста.

Квалификаторы типа информируют компилятор о необходимости и (или) возможности особой обработки объектов в процессе оптимизации кода программы.

Для обозначения классов памяти используются:

- **auto** – автоматический;
- **extern** – внешний;
- **register** – регистровый;
- **static** – статический.

Для построения операторов используются:

- **break** – завершить, прервать (например, цикл или переключатель);
- **case** – определяет вариант в операторе **switch**;
- **continue** – завершить текущую итерацию цикла (продолжить цикл, перейдя к следующей итерации);
- **default** – определяет действия при отсутствии нужного варианта в операторе **switch**;
- **do** – выполнять (заголовок оператора цикла с постусловием);

- **else** – входит в оператор **if**, определяя альтернативную ветвь;
- **for** – для (заголовок оператора параметрического цикла);
- **goto** – перейти (безусловный переход);
- **if** – «если» – обозначение условного оператора;
- **return** – возврат (из функции);
- **sizeof** – операция определения размера операнда (в байтах);
- **switch** – переключатель;
- **while** – «пока» (заголовок цикла с предусловием или завершение цикла **do**).

Конструкции языка, в которых используются служебные слова, будем определять по мере необходимости. Можно было бы не перечислять всех служебных слов, а вводить их по мере изложения языка, однако их запрещено использовать в качестве имен, выбираемых программистом, и поэтому для предупреждения возможных ошибок список служебных слов нужен уже на данном этапе.

Добавим еще одно соглашение, обычно соблюдаемое авторами компиляторов и стандартных библиотек языка Си. Идентификаторы, начинающиеся с одного или двух символов подчеркивания «_», зарезервированы для использования в библиотеках и компиляторах. Поэтому такие идентификаторы не рекомендуется выбирать в качестве имен в прикладной программе на языке Си. Следующее соглашение относительно имен относится уже не к стандарту и не к реализациям, а отображает стиль оформления текста программы. Рекомендуется при программировании имена констант записывать целиком заглавными буквами.

1.2. Литералы

По определению, константа представляет значение, которое не может быть изменено. Синтаксис языка определяет пять типов констант: символы, константы перечисляемого типа, вещественные числа, целые числа и нулевой указатель («null-указатель»). Все константы, кроме нулевого указателя, отнесены в языке Си к арифметическим.

Символы, или символьные константы. Для изображения отдельных знаков, имеющих индивидуальные внутренние коды, используются символьные константы. Каждая символьная константа – это лексема, которая состоит из изображения символа и ограничивающих апострофов. Например: 'A', 'a' , 'B' , '8' , '0' , '+' , ';' и т. д.

Внутри апострофов можно записать любой символ, изображаемый на дисплее или принтере в текстовом режиме. Однако в ЭВМ используются и коды, не имеющие графического представления на экране дисплея, клавиатуре или принтере. Примерами таких кодов служит код перехода курсора дисплея на новую строку или код возврата каретки (возврат курсора к началу текущей строки). Для изображения в программе соответствующих символьных констант используются комбинации из нескольких символов, имеющих графическое представление. Каждая такая комбинация начинается с символа '\' (обратная косая черта – backslash). Такие наборы литер, начинающиеся с символа '\', в литературе по языку Си называют управляющими последовательностями. Ниже приводится их список:

'\n' – перевод строки;
'\t' – горизонтальная табуляция;
'\r' – возврат каретки (курсора) к началу строки;
'\\' – обратная косая черта \;
'\'' – апостроф (одиночная кавычка);
'\"' – кавычка (символ двойной кавычки);
'\0' – нулевой символ;
'\a' – сигнал-звонок;
'\b' – возврат на одну позицию (на один символ);
'\f' – перевод (прогон) страницы;
'\v' – вертикальная табуляция;
'\?' – знак вопроса.

Обратите внимание на то, что перечисленные константы изображаются двумя и более литерами, а обозначают они одну символьную константу, имеющую индивидуальный двоичный код. Управляющие последовательности являются частным случаем эскейп-последовательностей (ESC-sequence), к которым также относятся лексемы вида '\ddd', '\xhh' или '\Xhh'.

'\ddd' – восьмеричное представление любой символьной константы. Здесь d – восьмеричная цифра (от 0 до 7). Например, '\017' или '\233'.

'\xhh' или '\Xhh' – шестнадцатеричное представление любой символьной константы. Здесь h – шестнадцатеричная цифра (от 0 до F). Например, '\x0b', '\x1A' и т. д.

В приложении 1 приведены таблицы числовых кодов символов. Зная значения кодов, можно изображать в виде эскейп-последова-

тельности любой символ, как воспроизводимый в текстовом режиме на дисплее (принтере), так и неизображаемый (управляющий).

Символьная константа (символ) имеет целый тип, то есть символы можно использовать в качестве целочисленных операндов в выражениях.

Целые константы. Синтаксисом языка определены целые константы: десятичные, шестнадцатеричные и восьмеричные. Основание определяется префиксом в записи константы. Для десятичных констант префикс не используется. Десятичные целые определены как последовательности десятичных цифр, начинающиеся не с нуля (если это не число нуль):

44 684 0 1024

Последовательность цифр, начинающаяся с 0 и не содержащая десятичных цифр старше 7, воспринимается как восьмеричная константа:

016 – восьмеричное представление десятичного целого 14.

Последовательность шестнадцатеричных цифр (0, 1, ..., 9, A, B, C, D, E, F), перед которой записаны символы 0x или 0X, считается шестнадцатеричной константой:

0x16 – шестнадцатеричное представление десятичного целого 22;
0XFF – шестнадцатеричное представление десятичного целого 255.

Каждая конкретная реализация языка вводит свои ограничения на предельные значения констант. Например, компилятор Turbo C в отношении целых констант соответствует стандарту и допускает целые десятичные от 0 до 32767, а длинные целые (см. ниже тип **long**) – от 0 до 2147483647.

Вещественные константы. Для представления вещественных (нецелых) чисел используются константы, представляемые в памяти ЭВМ в форме с плавающей точкой. Каждая вещественная константа состоит из следующих частей: целая часть (десятичная целая константа); десятичная точка; дробная часть (десятичная целая константа); признак показателя «e» или «E»; показатель десятичной степени (десятичная целая константа, возможно, со знаком). При записи констант с плавающей точкой могут опускаться целая или дробная часть (но не одновременно); десятичная точка или символ экспоненты с показателем степени (но не одновременно). Примеры констант с плавающей точкой:

44. 3.14159 44e0 .314159E1 0.0

Предельные значения и типы арифметических констант. Машинное представление (код) программы на языке Си предполагает, что каждая константа, введенная в программе, занимает в ЭВМ некоторый участок памяти. Размеры этого участка памяти и интерпретация его содержимого определяются типом соответствующей константы. В приложении 2 приведены допустимые стандартом предельные значения для разных типов данных. Почти все компиляторы отводят символьным константам (символам) по одному байту (восемь бит). Тем самым вводится ограничение на все разнообразие символьных констант – их внутренние коды должны находиться в диапазоне от 0 до 255. В языках многих стран мира не весь набор символов (букв и знаков) может быть представлен с помощью одного байта. В настоящее время ведется систематическая международная работа по созданию многобайтового универсального кода (unicode), в рамках которого можно представлять символы почти всех алфавитов. Однако рассмотрение многобайтовых кодов в рамках настоящего пособия представляется нам преждевременным.

Для целых и вещественных констант каждая реализация компилятора с языка Си может определять свои ограничения. Кроме того, необходимо учитывать платформу (аппаратная и программная составляющие), в среде которой реализован компилятор с языка Си, и платформу, в среде которой будет эксплуатироваться созданный программный продукт (см. приложение 4). В табл. 1.1 приведены пределы, исходя из которых компиляторы, реализованные на современных персональных компьютерах (ПК), выбирают типы целых констант. Например, все целые константы в диапазоне от 0 до 32767 имеют тип **int**, то есть будут представлены в памяти участками в 2 байта (16 бит).

Таблица 1.1. Целые константы и выбираемые для них типы

Диапазоны значений констант			Тип данных
десятичные	**восьмеричные**	**шестнадцатеричные**	
от 0 до 32767	от 00 до 077777	от 0x0000 до 0x7FFF	int
–	от 0100000 до 0177777	от 0x8000 до 0xFFFF	unsigned int
от 32768 до 2147483647	от 020000 до 017777777777	от 0x10000 до 0x7FFFFFFF	long
от 2147483648 до 4294967295	от 020000000000 до 037777777777	от 0x80000000 до 0xFFFFFFFF	unsigned long
до 9223372036854775807	до 0377777777777777777777	до 0x7FFFFFFFFFFFFFFF	long long
> 9223372036854775807	>0377777777777777777777	> 0x7FFFFFFFFFFFFFFF	ошибка

В табл. 1.2 приведены сведения о представлении данных вещественных типов.

Таблица 1.2. Данные вещественных типов

Тип данных	Размер, бит	Диапазон абсолютных величин
float	32	от 3.4E–38 до 3.4E+38
double	64	от 1.7E–308 до 1.7E+308
long double	80	от 3.4E–4932 до 1.1E+4932

Вещественная константа 3.141592653589793 будет воспринята как имеющая тип **double**, и ей будет выделено 8 байт (64 бита). Тот же тип выбирается для константы 3.14, так как по умолчанию всем вещественным константам присваивается тип **double**.

Если программиста не устраивает тип, который компилятор приписывает константе, то тип можно явно указать в записи константы с помощью суффиксов: **F** (или **f**) – **float** (для вещественных), **U** (или **u**) – **unsigned** (для целых), **L** (или **l**) – **long** (для целых и вещественных). Например:

3.14159F – константа типа **float** (выделяется 4 байта);
3.14L – константа типа **long double** (выделяется 10 байт).

С помощью суффикса **U** (или **u**) можно представить целую константу в виде беззнакового целого. Например:

50000U – константа типа **unsigned int**.

Константе 50000U выделяются 2 байта (вместо четырех, как было бы при отсутствии суффикса (см. табл. 1.1)). В этом случае, то есть для **unsigned int**, знаковый бит используется для представления одного из разрядов кода числа, и диапазон значений становится от 0 до 65535.

Суффикс **L** (или **l**) позволяет выделить целой константе 4 байта (32 бита):

500L – константа типа **long**, которой выделяется 4 байта;
0L – целая константа типа **long** длиной 4 байта.

Совместное использование в любом порядке суффиксов **U** (или **u**) и **L** (или **l**) позволяет приписать целой константе тип **unsigned long,** и она займет в памяти 32 разряда (бита), причем знаковый разряд будет использоваться для представления разряда кода (а не знака). Примеры:

0LU – целая константа типа **unsigned long** длиной 4 байта;

2424242424UL – константа типа **unsigned long**;
123LL – целая константа типа **long long**.

Нулевой указатель. Null-указатель, называемый нулевым указателем, – это единственная неарифметическая константа. Ее роль и функциональные возможности станут ясны при изучении аппарата указателей (см. следующие главы). В конкретных реализациях null-указатель может быть представлен либо как 0, либо как 0L, либо как именованная константа **NULL**. Здесь нужно отметить, что значение константы **NULL** не обязано быть нулем и имеет право не совпадать с кодом символа '0'.

Константы перечисляемого типа. Целочисленные именованные константы можно вводить с помощью перечисления:

enum *тип_перечисления {список_именованных_констант},*

где **enum** – служебное слово, вводящее перечисление; *тип_перечисления* – его название – необязательный произвольный идентификатор; *список_именованных_констант* – разделенная запятыми последовательность идентификаторов или именованных констант вида: *имя_константы=значение_константы.*

Примеры:

```
enum {ONE=1, TWO, THREE, FOUR};
enum DAY {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
          THURSDAY, FRIDAY, SATURDAY};
enum BOOLEAN {NO, YES};
```

Если в списке нет ни одного элемента со знаком '=', то значения констант начинаются с 0 и увеличиваются на 1 слева направо. Таким образом, NO равно 0, YES равно 1, SUNDAY имеет значение 0, и FRIDAY имеет значение 5. Именованная константа со знаком '=' получает соответствующее значение (ONE=1), а следующие за ней именованные константы без явных значений увеличиваются на 1 каждая. В нашем примере TWO равно 2, THREE равно 3, FOUR равно 4.

Строки, или строковые константы. Формально строки (в соответствии со стандартом) не относятся к константам языка Си, а представляют собой отдельный тип его лексем. Для них в литературе используется еще одно название – «строковые литералы». Строковая константа определяется как последовательность символов (см. выше символьные константы), заключенная в двойные кавычки (не в апострофы):

```
"Образец строки"
```

Среди символов строки могут быть эскейп-последовательности, то есть сочетания знаков, соответствующие неизображаемым символам, или символам, задаваемым их внутренними кодами. В этом случае, как и в представлениях отдельных символьных констант, их изображения начинаются с обратной косой черты ' \' :

```
"\n Текст \n разместится \n в 3-х строках дисплея"
```

Представления строковых констант в памяти ЭВМ подчиняются следующим правилам. Все символы строки размещаются подряд, и каждый символ (в том числе представленный эскейп-последовательностью) занимает ровно 1 байт. В конце записи строковой константы компилятор помещает символ '\0'.

Таким образом, количество байтов, выделяемое в памяти ЭВМ для представления значения строки, ровно на 1 больше, чем число символов в записи этой строковой константы:

```
"Эта строка занимает в памяти ЭВМ 43 байта."
"Строка в 18 байт."
```

При работе с символьной информацией нужно помнить, что длина константы 'F' равна 1 байту, а длина строки "F" равна 2 байтам.

При записи строковых констант возможно размещение одной константы в нескольких строках текстового файла с программой. Для этого используется следующее правило.

Если в последовательности символов (литер) константы встречается литера '\', за которой до признака '\n' конца строки текстового файла размещены только пробелы, то эти пробелы вместе с символом '\' и окончанием '\n' удаляются, и продолжением строковой константы считается следующая строка текста. Например, следующий текст представляет одну строковую константу:

```
"Шалтай-Болтай \
      сидел на стене."
```

В программе эта константа будет эквивалентна такой:

```
"Шалтай-Болтай      сидел на стене."
```

Начальные (левые) пробелы в продолжении константы на новой строке не удаляются, а считаются входящими в строковую константу.

Две строковые константы, между которыми нет других разделителей, кроме обобщенных пробельных символов (пробел, табуляция, конец строки и т. д.), воспринимаются как одна строковая константа. Таким образом,

```
"Шалтай-Болтай" " свалился во сне."
```

Воспринимается как одна константа:

```
"Шалтай-Болтай свалился во сне."
```

Тем же правилам подчиняются и строковые константы, размещенные на разных строках. Как одна строка будет воспринята последовательность

```
"Вся королевская "
                    "конница, "
  "вся королевская "
                    "рать"
```

Эти четыре строковые константы эквивалентны одной:

```
"Вся королевская конница, вся королевская рать"
```

Обратите внимание, что в результирующую строку здесь не включаются начальные пробелы перед каждой константой-продолжением.

1.3. Переменные и именованные константы

Переменная как объект. Одним из основных понятий языка Си является объект – именованная область памяти. Частный случай объекта – переменная. Отличительная особенность переменной состоит в возможности связывать с ее именем различные значения, совокупность которых определяется типом переменной. При задании значения переменной в соответствующую ей область памяти помещается код этого значения. Доступ к значению переменной наи-

более естественно обеспечивает ее имя, а доступ к участку памяти возможен только по его адресу. О взаимосвязях имен и адресов будет подробно говориться в главе, посвященной указателям и работе с памятью ЭВМ. Для целей первых глав будет вполне достаточно интерпретировать понятие переменной как пару «имя – значение».

Определение переменных. Каждая переменная перед ее использованием в программе должна быть определена, то есть для переменной должна быть выделена память. Размер участка памяти, выделяемой для переменной, и интерпретация содержимого зависят от типа, указанного в определении переменной.

В соответствии с типами значений, допустимых в языке Си, рассмотрим символьные, целые и вещественные переменные автоматической памяти. О классах памяти (один из которых – класс автоматической памяти) будем подробно говорить позже. Сейчас достаточно ввести только переменные автоматической памяти, которые существуют в том блоке, где они определены. В наиболее распространенном случае таким блоком является текст основной (main) функции программы.

Простейшая форма определения переменных:

тип список_имен_переменных,

где *имена переменных* – это выбранные программистом идентификаторы, которые в списке разделяются запятыми; *тип* – один из уже упоминаемых (в связи с константами) типов.

Определены целочисленные типы (перечислены в порядке неубывания длины внутреннего представления):

- **char** – целый длиной не менее 8 бит;
- **short int** – короткий целый (допустима аббревиатура **short**);
- **int** – целый;
- **long** – длинный целый.

Каждый из целочисленных типов может быть определен либо как знаковый **signed**, либо как беззнаковый **unsigned** (по умолчанию **signed**).

Различие между этими двумя типами – в правилах интерпретации старшего бита внутреннего представления. Спецификатор **signed** требует, чтобы старший бит внутреннего представления воспринимался как знаковый; **unsigned** означает, что старший бит внутреннего представления входит в код представляемого числового значения, которое считается в этом случае беззнаковым. Выбор знакового или беззнакового представления определяет предельные