Раздел 2. Язык программирования Си

Любая программа, написанная на языке высокого уровня, состоит из последовательности инструкций, оформленных в строгом соответствии с набором правил, составляющих ***синтаксис данного языка***.

При создании программ разработчик может допустить следующие ошибки: синтаксические и логические.

Синтаксические ошибки – это результат нарушения формальных правил написания программы на конкретном языке программирования.

Логические ошибки разделяются, в свою очередь, на ошибки алгоритма и семантические ошибки.

Причиной ошибки алгоритма является несоответствие построенного алгоритма ходу получения конечного результата сформулированной задачи.

Причина семантической ошибки – неправильное понимание смысла (семантики) операторов выбранного языка программирования.

2.1. Введение в язык Си. Структура программы

Алфавит любого языка составляет совокупность символов – тех неделимых знаков, при помощи которых записываются все тексты на данном языке.

Каждому из множества значений, определяемых одним байтом (от 0 до 255), в таблице знакогенератора ЭВМ ставится в соответствие символ. По кодировке фирмы *IBM* символы с кодами от 0 до 127, образующие первую половину таблицы знакогенератора, построены по стандарту *ASCII* и одинаковы для всех компьютеров, вторая половина символов (коды 128 – 255) может отличаться и обычно используется для размещения символов национального алфавита. Коды 176 – 223 отводятся под символы псевдографики, а коды 240 – 255 – под специальные знаки (прил. 1).

Алфавит языка Си включает:

– прописные и строчные буквы латинского алфавита и знак подчеркивания (код 95);

– арабские цифры от 0 до 9;

– специальные символы, смысл и правила использования которых будем рассматривать по тексту;

– пробельные (разделительные) символы: пробел, символы табуляции, перевода строки, возврата каретки, новой страницы и новой строки.

Лексемы

Из символов алфавита формируются *лексемы* (или элементарные конструкции) языка – минимальные значимые единицы текста в программе:

– идентификаторы;

– ключевые (зарезервированные) слова;

– знаки операций;

– константы;

– разделители (скобки, точка, запятая, пробельные символы).

Границы лексем определяются другими лексемами, такими как разделители или знаки операций, а также комментариями.

Идентификаторы и ключевые слова

Идентификатор (***ID***) – это имя программного объекта* (константы, переменной, метки, типа, функции и т.д.). В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания; первый символ *ID* – не цифра; пробелы внутри *ID* не допускаются.

Длина идентификатора определяется выбранной версией среды программирования. Например, в среде *Borland* C++ 6.0 идентификаторы могут включать любое число символов, из которых воспринимаются и используются только первые 32 символа. Современная тенденция – снятие ограничений длины идентификатора.

При именовании объектов следует придерживаться общепринятых соглашений:

– *ID* переменных и функций обычно пишутся строчными (малыми) буквами – *index, max*();

– *ID* типов пишутся с большой буквы, например, *Spis*, *Stack*;

– *ID* констант (макросов) – большими буквами – *INDEX*, *MAX_INT*;

– идентификатор должен нести смысл, поясняющий назначение объекта в программе, например, *birth_date* – день рождения, *sum* – сумма;

– если *ID* состоит из нескольких слов, как, например, *birth_date*, то принято либо разделять слова символом подчеркивания, либо писать каждое следующее слово с большой буквы – *birthDate*.

В Си прописные и строчные буквы – различные символы. Идентификаторы *Name, NAME, name* – различные объекты.

Ключевые (зарезервированные) слова не могут быть использованы в качестве идентификаторов.

Список ключевых слов, определенных в стандарте ANSI Си:

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

* Здесь и далее по тексту объектами будем называть элементы, участвующие в программе.

Комментарии

Еще один базовый элемент языка программирования – *комментарий* – не является лексемой. Внутри комментария можно использовать любые допустимые на данном компьютере символы, поскольку компилятор их игнорирует.

В Си комментарии ограничиваются парами символов /* и */, а в C++ был введен вариант комментария, который начинается символами // и заканчивается символом перехода на новую строку.

Простейшая программа

Программа, написанная на языке Си, состоит из одной или нескольких функций, одна из которых имеет идентификатор ***main***[*] – главная (основная). Она является первой выполняемой функцией (с нее начинается выполнение программы) и ее назначение – управлять работой всей программы (проекта).

Общая структура программы на языке Си имеет вид:

<директивы препроцессора>
<определение типов пользователя – *typedef*>
<описание прототипов функций>
<определение глобальных переменных>
<функции>

В свою очередь, каждая функция имеет следующую структуру:

<класс памяти> <тип> < *ID* функции> (<объявление параметров>)
{ – начало функции
код функции
} – конец функции

Код функции является блоком и поэтому заключается в фигурные скобки.

Функции не могут быть вложенными друг в друга.

Рассмотрим кратко основные части общей структуры программ.

Перед компиляцией программа обрабатывается ***препроцессором*** (прил. 3), который работает под управлением директив.

Препроцессорные ***директивы*** начинаются символом #, за которым следует наименование директивы, указывающее ее действие.

Препроцессор решает ряд задач по предварительной обработке программы, основной из которых является подключение (***include***) к программе так называемых заголовочных файлов (обычных текстов) с декларацией стандартных библиотечных функций, использующихся в программе. Общий формат ее использования

#include < ID_файла.h>

где *h* – расширение заголовочных файлов.

[*] Более подробное описание функции *main* рассматривается в п. 11.7.

Если идентификатор файла заключен в угловые скобки (< >), то поиск данного файла производится в стандартном каталоге, если – в двойные кавычки (” ”), то поиск файла производится в текущем каталоге.

К наиболее часто используемым библиотекам относятся:

stdio.h – содержит стандартные функции файлового ввода-вывода;

math.h – математические функции;

conio.h – функции для работы с консолью (клавиатура, дисплей).

Второе основное назначение препроцессора – обработка макроопределений. Макроподстановка ***определить*** (***define***) имеет общий вид

#define ID строка

Например: *#define PI* 3.1415927

– в ходе препроцессорной обработки программы идентификатор *PI* везде будет заменяться значением 3.1415927.

Рассмотрим пример, позволяющий понять простейшие приемы программирования на языке Си:

```
#include <stdio.h>
void main(void)
{                          // Начало функции main
    printf(“ Высшая оценка знаний – 10 !”);
}                          // Окончание функции main
```

Отличительным признаком функции служат скобки ( ) после ее идентификатора, в которые заключается список параметров. Перед *ID* функции указывается тип возвращаемого ею результата. Если функция не возвращает результата и не имеет параметров, указывают атрибуты *void* – отсутствие значений.

Для начала будем использовать функцию *main* без параметров и не возвращающую значения.

Код функции представляет собой набор инструкций, каждая из которых оканчивается символом «**;**». В нашем примере одна инструкция – функция *printf*, выполняющая вывод данных на экран, в данном случае – указанную фразу.

Приемы отладки в среде программирования *Visual* C++ 6.0 рассматриваются в прил. 5.

Основные типы данных

Данные в языке Си разделяются на две категории: простые (скалярные), будем их называть базовыми, и сложные (составные) типы данных.

Тип данных определяет:

– внутреннее представление данных в оперативной памяти;

– совокупность значений (диапазон), которые могут принимать данные этого типа;

– набор операций, которые допустимы над такими данными.

Основные типы базовых данных: целый – ***int*** (*integer*), вещественный с одинарной точностью – ***float*** и символьный – ***char*** (*character*).

В свою очередь, данные целого типа могут быть короткими – ***short***, длинными – ***long*** и беззнаковыми – ***unsigned***, а вещественные – с удвоенной точностью – ***double***.

Сложные типы данных – массивы, структуры – ***struct***, объединения – ***union***, перечисления – ***enum***.

Данные целого и вещественного типов находятся в определенных диапазонах, т.к. занимают разный объем оперативной памяти (табл. 2.1).

Таблица 2.1

Тип данных	Объем памяти (байт)	Диапазон значений
char	1	–128 … 127
int	2 (4)*	–32768 … 32767
short	1 (2)*	–32768 … 32767(–128 … 127)
long	4	–2147483648 … 2147483647
unsigned int	4	0 … 65535
unsigned long	4	0 … 4294967295
float	4	$3{,}14\cdot10^{-38}$ … $3{,}14\cdot10^{38}$
double	8	$1{,}7\cdot10^{-308}$ … $1{,}7\cdot10^{308}$
long double	10	$3{,}4\cdot10^{-4932}$ … $3{,}4\cdot10^{4932}$

* Размер памяти зависит от разрядности процессора, для 16-разрядных объем памяти определяется первой цифрой, для 32-разрядных – второй.

Декларация объектов

Все объекты, с которыми работает программа, необходимо декларировать, т.е. объявлять компилятору об их присутствии. При этом возможны две формы декларации:

– описание, не приводящее к выделению памяти;

– определение, при котором под объект выделяется объем памяти в соответствии с его типом; в этом случае объект можно инициализировать, т.е. задать его начальное значение.

Кроме констант, заданных в исходном тексте, все объекты программы должны быть явно декларированы по следующему формату:

<атрибуты> <список ID объектов>;

элементы *списка ID объектов* разделяются запятыми, а *атрибуты* – разделителями, например: *int i, j, k*; *float a, b*;

Объекты программы могут иметь следующие атрибуты:

класс памяти – характеристика способа размещения объектов в памяти (статическая, динамическая); определяет область видимости и время жизни переменной (по умолчанию – ***auto***), данные атрибуты будут рассмотрены в гл. 12;

тип – тип будущих значений декларируемых объектов (по умолчанию устанавливается тип ***int***).

Класс памяти и тип – атрибуты необязательные и при отсутствии одного из них (но не обоих одновременно) устанавливаются атрибуты по умолчанию.

Примеры декларации простых объектов:

int i, j, k; *char r;* *double gfd;*

Рассмотрим основные базовые типы данных более подробно.

Данные целого типа (integer)

Тип ***int*** – целое число, обычно соответствующее естественному размеру целых чисел. Квалификаторы ***short*** и ***long*** указывают на различные размеры и определяют объем памяти, выделяемый под них (см. табл. 2.1), например:

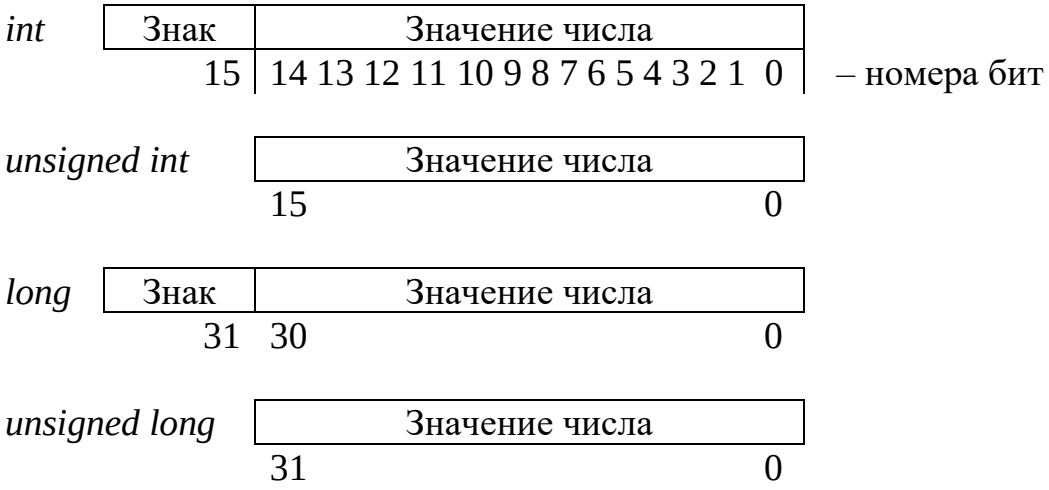
short x;

long x;

unsigned x = 8; – декларация с инициализацией числом 8;

атрибут *int* в этих случаях может быть опущен.

Атрибуты ***signed*** и ***unsigned*** показывают, как интерпретируется старший бит числа – как знак или как часть числа:



Данные символьного типа (char)

Под величину символьного типа отводится такое количество байт, которое достаточно для любого символа. Поэтому символьная переменная занимает в памяти один байт. Закрепление конкретных символов за кодами производится кодовыми таблицами.

Для персональных компьютеров (ПК) наиболее распространена *ASCII* (*American Standard Code for Information Interchenge*) таблица кодов (см. прил. 1). Данные типа *char* рассматриваются компилятором как целые, поэтому возможно использование *signed char*: величины со знаком (по умолчанию) – символы с кодами от –128 до +127 и *unsigned char* – беззнаковые символы с кодами от 0 до 255. Этого достаточно для хранения любого символа из 256-символьного набора *ASCII*. Величины типа *char* применяют еще и для хранения целых чисел из указанных диапазонов.

Примеры: *char res, simv*1, *simv*2;

char let = 's'; – декларация символьной переменной с инициализацией символом *s*.

Данные вещественного типа (float, double)

Данные вещественного типа в памяти занимают (табл. 2.2): ***float*** – 4 байта (одинарная точность), ***double*** (удвоенная точность) – 8 байт; ***long double*** (повышенная точность) – 10 байт. Для размещения данных типа *float* обычно 8 бит выделено для представления порядка и знака и 24 бита под мантиссу.

Таблица 2.2

Тип	Точность (мантисса)	Порядок
float (4 байта)	7 цифр после запятой	± 38
double (8 байт)	15	± 308
long double (10 байт)	19	± 4932

Типы данных с плавающей десятичной точкой хранятся в оперативной памяти иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей: мантиссы и порядка (см. разд. 3.2 «Константы вещественного типа»). В *IBM* совместимых ПК, как вы уже знаете, переменная типа *float* занимает 4 байта, из которых один двоичный разряд отводится под знак мантиссы, 8 разрядов под порядок и 23 под мантиссу. Мантисса – это число больше единицы и меньше двух. Поскольку старшая цифра мантиссы всегда равна единице, то ее не хранят.

Для величин типа *double*, занимающих 8 байт, под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а порядок – его диапазон. Как можно видеть из приведенных выше таблиц, при одинаковом количестве байт, отводимом под величины типа *float* и *long int*, диапазоны их допустимых значений сильно различаются из-за внутренней формы представления значений таких данных.

При переносе программы с одной платформы на другую нельзя делать предположений, например, о типе *int*, так как для оперативной системы (ОС) *MS DOS* этот тип имеет размер в два байта, а для ОС *Windows* 9X – четыре байта. В стандарте *ANSI* поэтому диапазоны значений для основных типов не задаются, а определяются только соотношения между их размерами, например:

sizeof (***float***) < *sizeof* (***double***) < *sizeof* (***long double***) ,
sizeof (***char***) < *sizeof* (***short***) < *sizeof* (***int***) < *sizeof* (***long***) ,

где операция ***sizeof*** – возвращает количество байт для указанного аргумента – скалярного типа данных.

Использование модификаторов при декларации производных типов данных

Ключевые слова *int, float, char* и т.д. называют конечными атрибутами декларации объектов программы. При декларации так называемых производных

объектов используют еще дополнительные – промежуточные атрибуты или, как их иногда называют, «***модификаторы***».

К символам модификации текущего типа относятся:

– символ ***** перед идентификатором, обозначающий декларацию указателя на объект исходного типа (левый промежуточный атрибут);

– символы [ ] после идентификатора объекта – декларация массива объектов;

– символы ( ) после идентификатора объекта – декларация функции (правые промежуточные атрибуты).

Допускается использование более одного модификатора типа с учетом следующих правил:

1) чем ближе модификатор к *ID* объекта, тем выше его приоритет;

2) при одинаковом расстоянии от идентификатора объекта модификаторы [ ] и ( ) обладают приоритетом перед атрибутом звездочка *;

3) дополнительные круглые скобки позволяют изменить приоритет объединяемых ими элементов описания;

4) квадратные и круглые скобки, имеющие одинаковый приоритет, рассматриваются слева направо.

Конечный атрибут декларации принимается во внимание в последнюю очередь, т.е. тогда, когда все промежуточные атрибуты уже проинтерпретированы.

Примеры декларации объектов с конечным атрибутом ***int***:

int a; – переменная типа *int*;

int a[5]; – массив из пяти элементов типа *int*;

*int *a*; – указатель на объект типа *int*;

*int **a*; – указатель на указатель на объект типа *int*;

*int *a*[5]; – массив из пяти указателей на элементы типа *int*;

int (**a*)[10]; – указатель на массив из десяти элементов типа *int*;

*int *a*[3][4]; – 3-элементный массив указателей на одномерные целочисленные массивы по четыре элемента каждый;

int a[5][2]; – двухмерный массив элементов типа *int*;

int a(*void*); – функция без параметров, возвращающая значение типа *int*;

*int *a*(*void*); – функция без параметров, возвращающая указатель на элемент типа *int*;

int (**a*)(*void*); – указатель на функцию без параметров, возвращающую значение типа *int*;

*int *a*(*void*)[6]; – функция без параметров, возвращающая указатель на массив элементов типа *int*;

*int *a* [4](*void*); – массив указателей на функцию без параметров, возвращающую значение типа *int*.

Существуют и недопустимые последовательности промежуточных атрибутов, например, массив не может состоять из функций, а функция не может возвращать массив или другую функцию.

Константы в программах

Константами называют величины, которые не изменяют своего значения во время выполнения программы, т.е. это объекты, не подлежащие использованию в левой части операции присваивания, т.к. константа – это неадресуемая величина и, хотя она хранится в памяти компьютера, не существует способа определить ее адрес. В языке Си константами являются:

– самоопределенные арифметические константы целого и вещественного типов, символьные и строковые данные;

– идентификаторы массивов и функций;

– элементы перечислений.

Целочисленные константы

Общий формат записи: **±*n*** (+ обычно не ставится).

Десятичные константы – это последовательность цифр 0...9, первая из которых *не должна быть 0*. Например, 22 и 273 – обычные целые константы, если нужно ввести длинную целую константу, то указывается признак *L*(*l*) – 273*L* (273*l*). Для такой константы будет отведено – 4 байта. Обычная целая константа, которая слишком длинна для типа *int*, рассматривается как *long*.

Существует система обозначений для восьмеричных и шестнадцатеричных констант.

Восьмеричные константы – это последовательность цифр от 0 до 7, первая из которых *должна быть 0*, например: $020_8 = 16_{10}$.

Шестнадцатеричные константы – последовательность цифр от 0 до 9 и букв от *A* до *F* (*a*...*f*), начинающаяся символами 0*X* (0*x*), например: $0X1F_{16}$ $(0x1f)_{16} = 31_{10}$.

Восьмеричные и шестнадцатеричные константы могут также заканчиваться буквой *L*(*l*) – *long*, например, 020*L* или 0*X*20*L*.

Примеры целочисленных констант:

1992	777	1000*L*	– десятичные;
0777	00033	01*l*	– восьмеричные;
0*x*123	0*X*00*ff*	0*xb*8000*l*	– шестнадцатеричные.

Константы вещественного типа

Данные константы размещаются в памяти в формате *double*, а во внешнем представлении могут иметь две формы:

1) с фиксированной десятичной точкой, формат записи: **±*n.m***, где *n*, *m* – целая и дробная части числа;

2) с плавающей десятичной точкой (экспоненциальная форма) представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты (*E* или *e*), а порядок – справа. Значение константы определяется как произведения мантиссы и числа 10, возведенного в указанную в порядке степень.

Общий формат таких констант: ***±n.mE±p***, где *n, m* – целая и дробная части числа, *p* – порядок; ±0.*xxxE*±*p* – нормализованный вид, например, $1{,}25 \cdot 10^{-8}$ = 0.125*E*–7.

Примеры констант с фиксированной и плавающей точками:

1.0 –3.125 100*e*–10 0.12537*e*+12.

Пробелы внутри чисел не допускаются, а для отделения целой части числа от дробной используется *точка*. Можно опустить нулевую дробную или целую части числа, но не обе сразу, например, 1.0 ↔ 1. или 0.5 ↔ .5 .

В любом случае при использовании вещественных констант наличие так называемой десятичной точки обязательно.

Символьные константы

Символьная константа – это символ, заключенный в одинарные кавычки: '*A*', '*x*' (тип *char* занимает в памяти один байт).

Также используются специальные последовательности символов – управляющие (*escape*) последовательности:

\n – новая строка;
\t – горизонтальная табуляция;
\b – шаг назад;
\r – возврат каретки;
\v – вертикальная табуляция;
\f – перевод формата (переход на новую строку);
\\ – обратный слеш;
\' – апостроф;
\" – кавычки;
\0 – символ «пусто», не путать с символом ‘0’.

Символьная константа '\0' – это нулевой байт, каждый бит которого равен нулю.

При присваивании символьным переменным значений констант значения констант заключаются в апострофы, например:

char ss = ‘У’;

Текстовые символы непосредственно вводятся с клавиатуры, а специальные и управляющие – представляются в исходном тексте парами символов, например: \\ , \' , \" .

Примеры символьных констант: '*A*', '9', '$', '*n*'.

Строковые константы

Строковая константа представляет собой последовательность символов кода *ASCII*, заключенную в кавычки (”). Во внутреннем представлении к строковым константам добавляется пустой символ '\0', который не является цифрой 0, на печать не выводится (в таблице кодов *ASCII* имеет код = 0) и является признаком окончания строки.

Кавычки не являются частью строки, а служат только для ее ограничения. Строка в языке Си представляет собой массив, состоящий из символов. Внутреннее представление константы "1234*ABC*": '1' '2' '3' '4' '*A*' '*B*' '*C*' '\0' .

Примеры строковых констант:

"Система", "*n**t* Аргумент *n*", "Состояние \"*WAIT* \" " .

Строковые константы еще называют *строковыми литералами*.

В конец строковой константы компилятор автоматически помещает нуль-символ.

Длинную строковую константу можно разбить на несколько, используя символ переноса – обратный слеш (\). Например:

"*Вы поступили и* \
учитесь на факультете информационных технологий \
Белорусского государственного университета \
информатики и радиоэлектроники"

Компилятор Си воспримет такую запись как единое целое, игнорируя символы обратного слеша.