Chapter 4

Lexical Conventions

This chapter covers the C lexical conventions including comments and tokens. A token is a series of contiguous characters that the compiler treats as a unit. The classes of tokens described in the sections below include:

- "Identifiers"
- "Keywords"
- "Constants"
- "String Literals"
- "Operators"
- "Punctuators"

Blanks, tabs, new−lines, and comments (described in the next section) are collectively known as "white space." White space is ignored except as it serves to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters that could possibly constitute a token.

Comments

The characters **/*** introduce a comment; the characters ***/** terminate a comment. They do not indicate a comment when occurring within a string literal. Comments do not nest. Once the **/*** introducing a comment is seen, all other characters are ignored until the ending ***/** is encountered.

Identifiers

An identifier, or name, is a sequence of letters, digits, and underscores (_). The first character cannot be a digit. Uppercase and lowercase letters are distinct. Name length is unlimited. The terms *identifier* and *name* are used interchangeably.

Keywords

The identifiers listed in Table 4−1are reserved for use as keywords and cannot be used for any other purpose.

Table 4−1 Reserved Keywords

Keywords					
auto	default	float	register	struct	volatile
break	do	for	return	switch	while
case	double	goto	short	typedef	
char	else	if	signed	union	

const	enum	int	sizeof	unsigned
continue	extern	long	static	void

Traditional C reserves and ignores the keyword fortran.

Constants

The four types of constants are *integer*, *character*, *floating*, and *enumeration*. Each constant has a type, determined by its form and value.

In the following discussion of the various types of constants, a unary operator preceding the constant is not considered part of it. Rather, such a construct is a *constant−expression* (see "Constant Expressions"). Thus, the integer constant *0xff* becomes an integral constant expression by prefixing a minus sign, as *−0xff* The effect of the operator **-** is not considered in the discussion of integer constants.

As an example, the integer constant *0xffffffff* has type **int** in traditional C, with value −1. It has type **unsigned** in ANSI C, with value 232-1. This discrepancy is inconsequential if the constant is assigned to a variable of integral type (for example, **int** or **unsigned**), as a conversion occurs. If it is assigned to a **double**, however, the value differs as indicated between traditional and ANSI C.

Integer Constants

An integer constant consisting of a sequence of digits is considered octal if it begins with **0**. An octal constant consists of the digits **0** through **7** only. A sequence of digits preceded by **0x** or **0X** is considered a hexadecimal integer. The hexadecimal digits include **[aA]** through **[fF]** with values 10 through 15.

The suffixes [lL] traditionally indicate integer constants of type long. These suffixes are allowed, but are superfluous, since int and **long** are the same size in **-32** mode. The suffices **ll**, **LL**, **lL**, and **Ll** indicate a **long long** constant (a 64−bit integral type). Note that **long long** is not a strict ANSI C type, and a warning is given for **long long** constants in **-ansi** and **-ansiposix** modes. Examples of **long long** include:

```
12345LL
12345ll
```

In ANSI C, an integer constant can be suffixed with **[uU]**, in which case its type is **unsigned**. (One or both of **[uU]** and **[lL]** can appear.) An integer constant also has type **unsigned** if its value cannot be represented as an **int**. Otherwise, the type of an integer constant is **int**. Examples of unsigned **long long** include:

```
123456ULL
123456ull
```

Character Constants

A character constant is a character enclosed in single quotes, as in **'x'**. The value of a character constant is the numerical value of the character in the machine's character set. An explicit new−line character is illegal in a character constant. The type of a character constant is **int**.

In ANSI C, a character constant can be prefixed by **L**, in which case it is a wide character constant. For example, a wide character constant for **‘z’** is written **L’z’**. The type of a wide character constant is **wchar_t**, which is defined in *<stddef.h>*.

Special Characters

Some special and nongraphic characters are represented by the escape sequences shown in Table 4−2

Table 4−2 Escape Sequences for Nongraphic Characters

Character Name	Escape Sequence
new−line	\n
horizontal tab	\t
vertical tab	\v
backspace	\b
carriage return	\r
form feed	\f
backslash	\\
single quote	\’
double quote	\"
question mark	\?
bell (ANSI C only)	\a

The escape *ddd* consists of the backslash followed by 1, 2, or 3 octal digits that are taken to specify the value of the desired character. A special case of this construction is **\0** (not followed by a digit), which indicates the ASCII character **NUL**.

In ANSI C, **\x** indicates the beginning of a hexadecimal escape sequence. The sequence is assumed to continue until a character is encountered that is not a member of the hexadecimal character set 0,1,… 9, [aA], [bB], … [fF]. The resulting unsigned number cannot be larger than a character can accommodate (decimal 255).

If the character following a backslash is not one of those specified in this discussion, the behavior is undefined.

Trigraph Sequences (ANSI C Only)

The character sets of some older machines lack certain members that have come into common usage. To allow the machines to specify these characters, ANSI C defined an alternate method for their specification, using sequences of characters that are commonly available. These sequences are termed *trigraph sequences.* Nine sequences are defined, each consists of three characters beginning with two question marks. Each instance of one of these sequences is translated to the corresponding single character. Other sequences of characters, perhaps including multiple question marks, are unchanged. Each trigraph sequence with the single character it represents is listed in Table 4−3

Table 4−3 Trigraph Sequences

Trigraph Sequence	Single Character
??=	#
??(	[

??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??–	~

Floating Constants

A floating constant consists of an integer part, a decimal point, a fraction part, an [eE], and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (but not both) can be missing. Either the decimal point or the [eE] and the exponent (not both) can be missing.

In traditional C, every floating constant has type double.

In ANSI C, floating constants can be suffixed by either [fF] or [lL]. Floating constants suffixed with [fF] have type float. Those suffixed with [lL] have type long double, which has greater precision than double in **-64** mode and a precision equal to **double** in **-32** mode.

Enumeration Constants

Names declared as enumerators have type **int**. For a discussion of enumerators, see "Enumeration Declarations". For information on the use of enumerators in expressions, see "Integer and Floating Point Types".

String Literals

A string literal is a sequence of characters surrounded by double quotes, as in **"..."**. A string literal has type array of char and is initialized with the given characters. The compiler places a null byte (**\0**) at the end of each string literal so that programs that scan the string literal can find its end. A double–quote character (**"**) in a string literal must be preceded by a backslash (****). In addition, the same escapes as described for character constants can be used. (See "Character Constants" for a list of escapes.) A backslash (****) and the immediately following new line are ignored. Adjacent string literals are concatenated.

In traditional C, all string literals, even when written identically, are distinct.

In ANSI C, identical string literals are not necessarily distinct. Prefixing a string literal with **L** specifies a wide string literal. Adjacent wide string literals are concatenated.

As an example, consider the sentence *He said, "Hi there."* This sentence could be written with three adjacent string literals as

```
"He said, " "\"Hi " "there.\""
```

Operators

An *operator* specifies an operation to be performed. The operators [ ], ( ), and ? : must occur in pairs, possibly separated by expressions. The operators # and ## can occur only in preprocessing directives.

operator: one of

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

Individual operations are discussed in Chapter 7, "Expressions and Operators."

Punctuators

A *punctuator* is a symbol that has semantic significance but does not specify an operation to be performed. The punctuators [ ], ( ), and { } must occur in pairs, possibly separated by expressions, declarations or statements. The punctuator # can occur only in preprocessing directives.

punctuator: one of

[ ] ( ) { } * , : = ; ... #

Some operators, determined by context, are also punctuators. For example, the array index indicator [ ] is a punctuator in a declaration (see Chapter 8, "Declarations"), but an operator in an expression (see Chapter 7, "Expressions and Operators").