Unit 2: Basic Elements of C

Agendas

- C Standards
- C Character Set
- C Tokens
- Escape sequence
- Delimiters
- Variables,
- Data types
- Constants/Literals
- Expressions
- Statements and Comments
- Library Functions
- Preprocessor Directives.

C standards

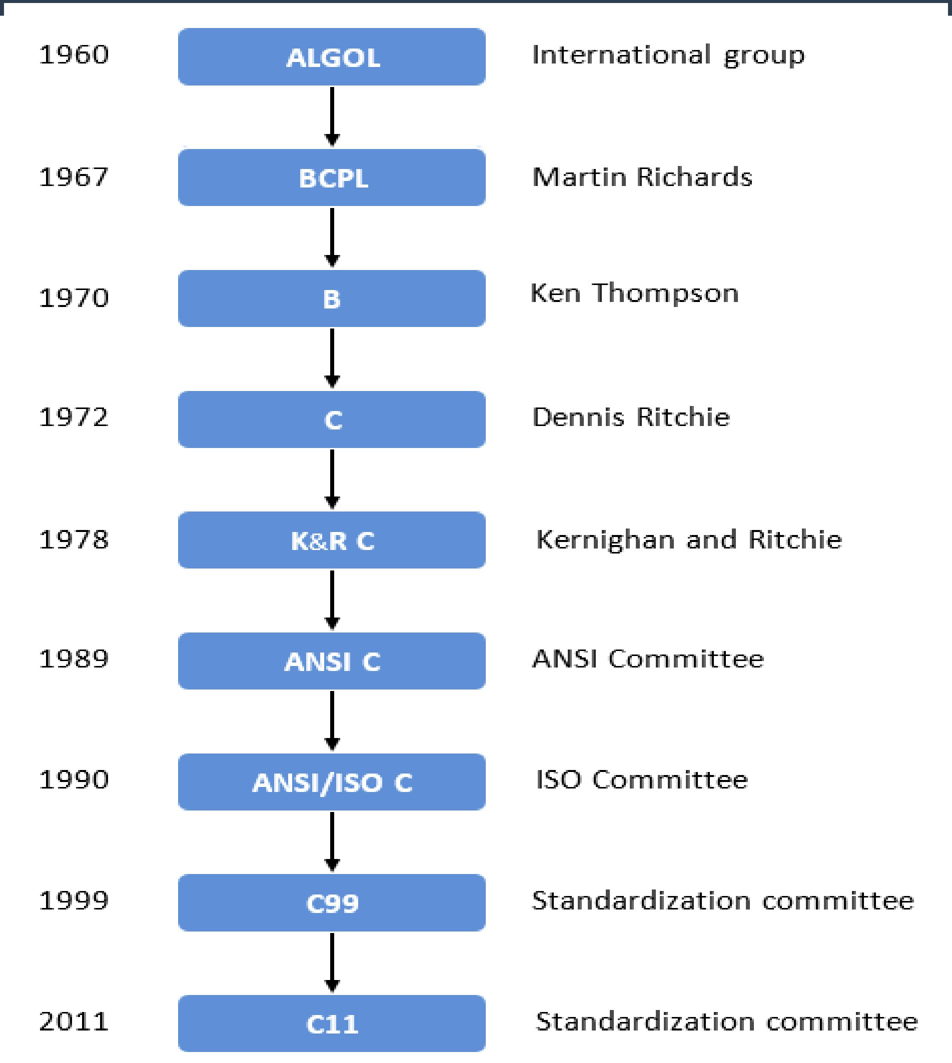
The massive popularity of C programming headed development of its versions. However, every version was similar to the original but often incompatible. To assure the standard of C in every version, American National Standards Institute (ANSI) initiated work on C standards.

In 1989, ANSI sets base for all implementations of C compilers and published the first standard of C. The first C standard is popularly known as C89. The current standard is C17.

C standards

```
//to check for C standard

printf("%d", __STDC_VERSION__);
```



1960 ALGOL International group
1967 BCPL Martin Richards
1970 B Ken Thompson
1972 C Dennis Ritchie
1978 K&R C Kernighan and Ritchie
1989 ANSI C ANSI Committee
1990 ANSI/ISO C ISO Committee
1999 C99 Standardization committee
2011 C11 Standardization committee

C89/C90

C95,

C99, (gets() deprecated) (added Boolean data type)

C11, (removed gets())

C17,

C23.

C standards

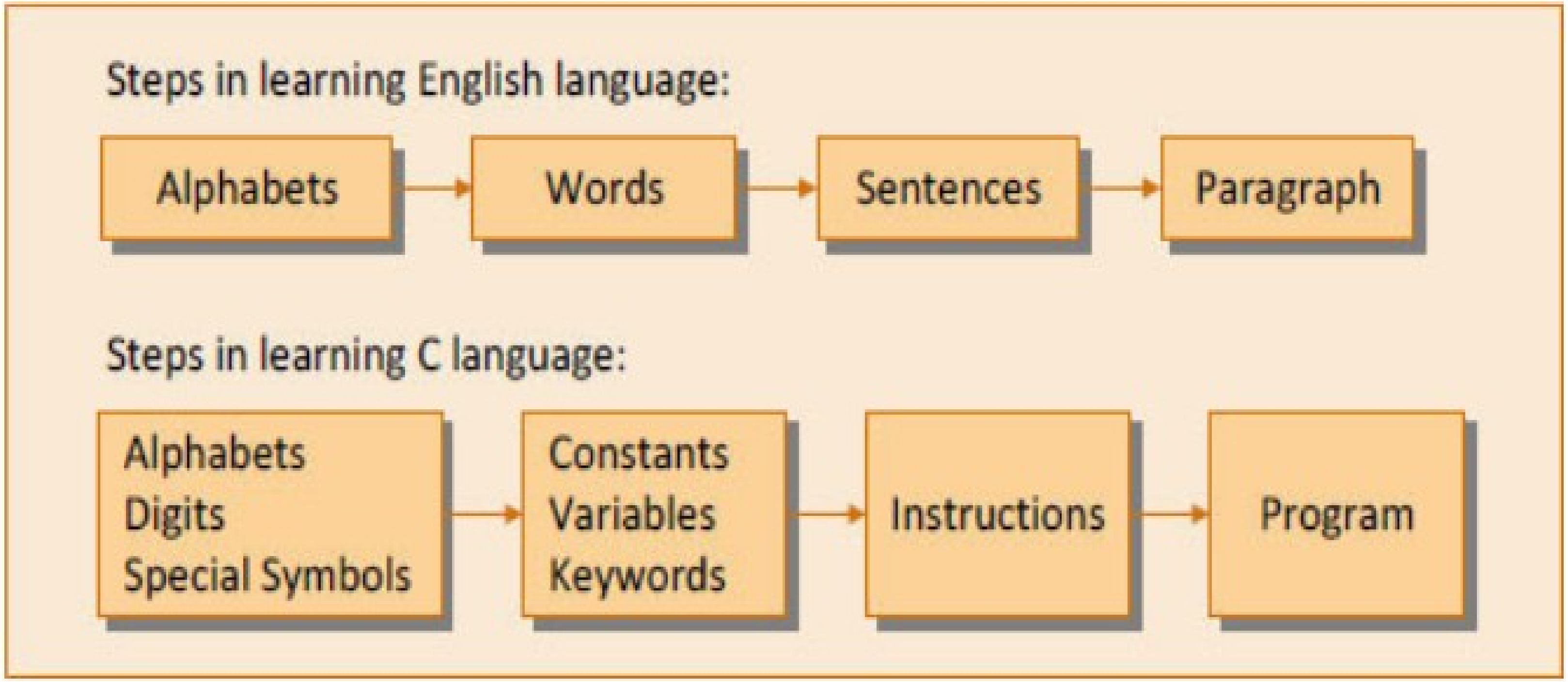
The C standards are official documents that are intended to reduce variation in C language implementation across the different compilers. They are highly technical documents and are written for compiler implementation.

C Character Set

- ✓The programming languages are designed to support certain kind of data, such as numbers, characters, strings, etc to get useful output known as result/information. Data consists of **digits, alphabets and symbols.**
- ✓A program should support these data types for getting the required output known as **information**.
- ✓A **program** is a set of statements, which performs a specific task, executed by users in a sequential form.
- ✓These statements/instructions are formed using certain words and symbols according to the rules known as **syntax or grammar** of the language. Every program must accurately follow the syntax rules supported by the language.

C Character

Comparison of English Language and C programming Language.


Steps in learning English language:
Alphabets
Words
Sentences
Paragraph
Steps in learning C language:
Alphabets Digits Special Symbols
Constants Variables Keywords
Instructions
Program

C Character

- ✓ **Characters can be used to form words, numbers, statements and expressions depending upon the computer on which the program runs and set of these characters are known as character set .**

- ✓ **C has its own characters set used to form words, statements and expressions.**

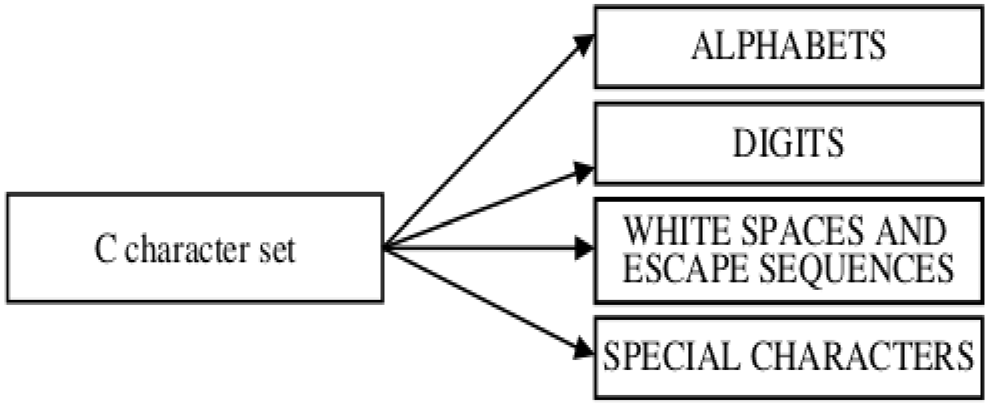
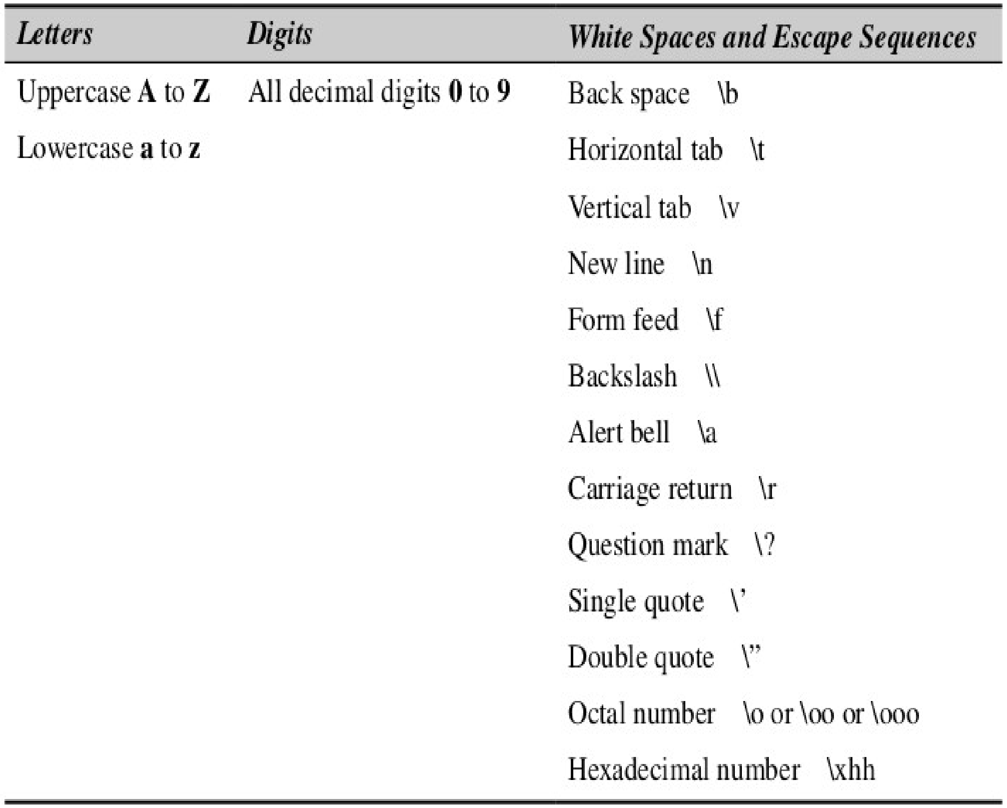


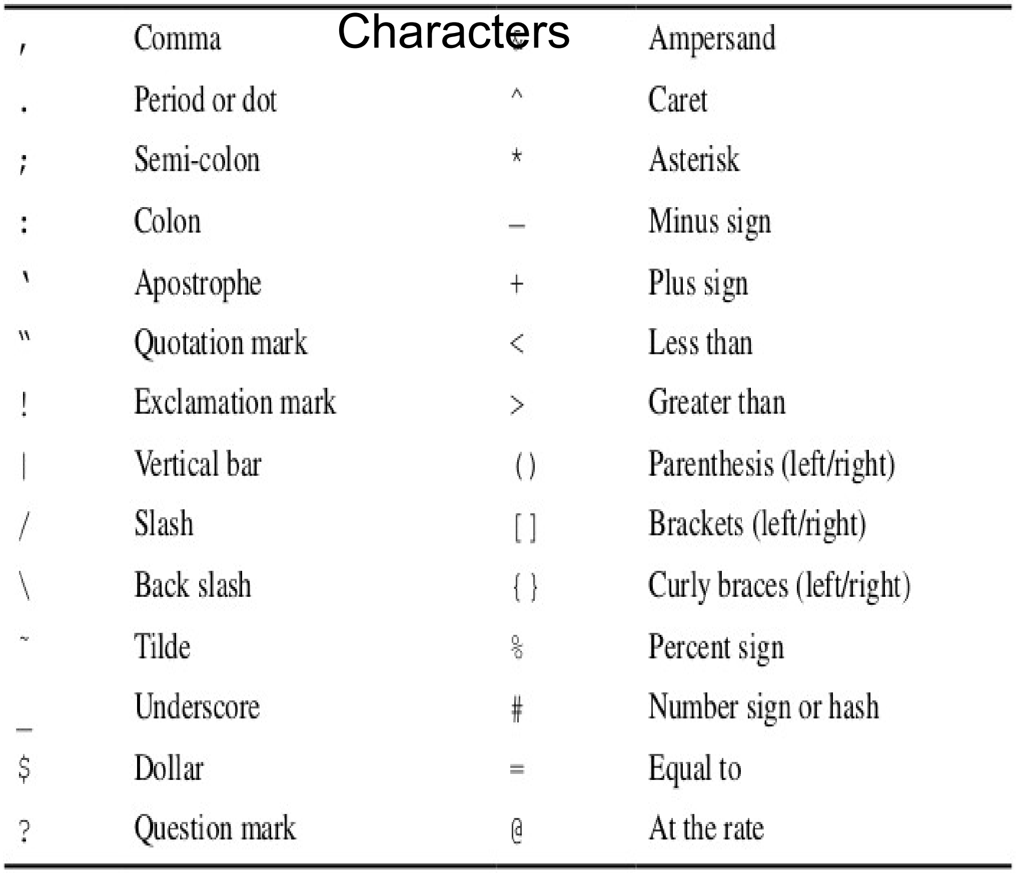
Figure: C character set

C Character set



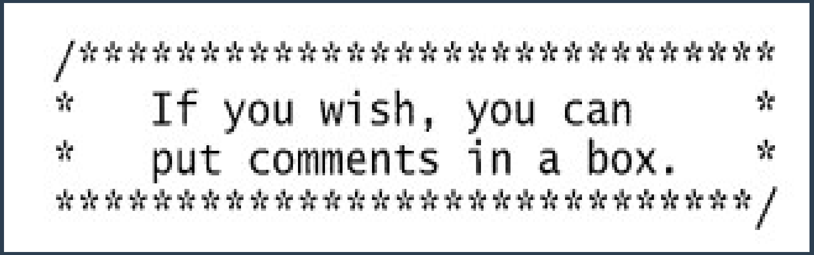
Letters	*Digits*	*White Spaces and Escape Sequences*
Uppercase **A** to **Z**	All decimal digits **0** to **9**	Back space \b
Lowercase **a** to **z**		Horizontal tab \t
		Vertical tab \v
		New line \n
		Form feed \f
		Backslash \\
		Alert bell \a
		Carriage return \r
		Question mark \?
		Single quote \'
		Double quote \"
		Octal number \o or \oo or \ooo
		Hexadecimal number \xhh

Special Characters



,	Comma	&	Ampersand
.	Period or dot	^	Caret
;	Semi-colon	*	Asterisk
:	Colon	–	Minus sign
'	Apostrophe	+	Plus sign
"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	( )	Parenthesis (left/right)
/	Slash	[ ]	Brackets (left/right)
\	Back slash	{ }	Curly braces (left/right)
~	Tilde	%	Percent sign
_	Underscore	#	Number sign or hash
$	Dollar	=	Equal to
?	Question mark	@	At the rate

Comments in C



```
/******************************
*   If you wish, you can      *
*   put comments in a box.    *
******************************/
```

- Comments are arbitrary strings of symbols placed between the delimiters /* and */ used as documenting aid for codes/instructions. Example:

  /* This is multiple line comment

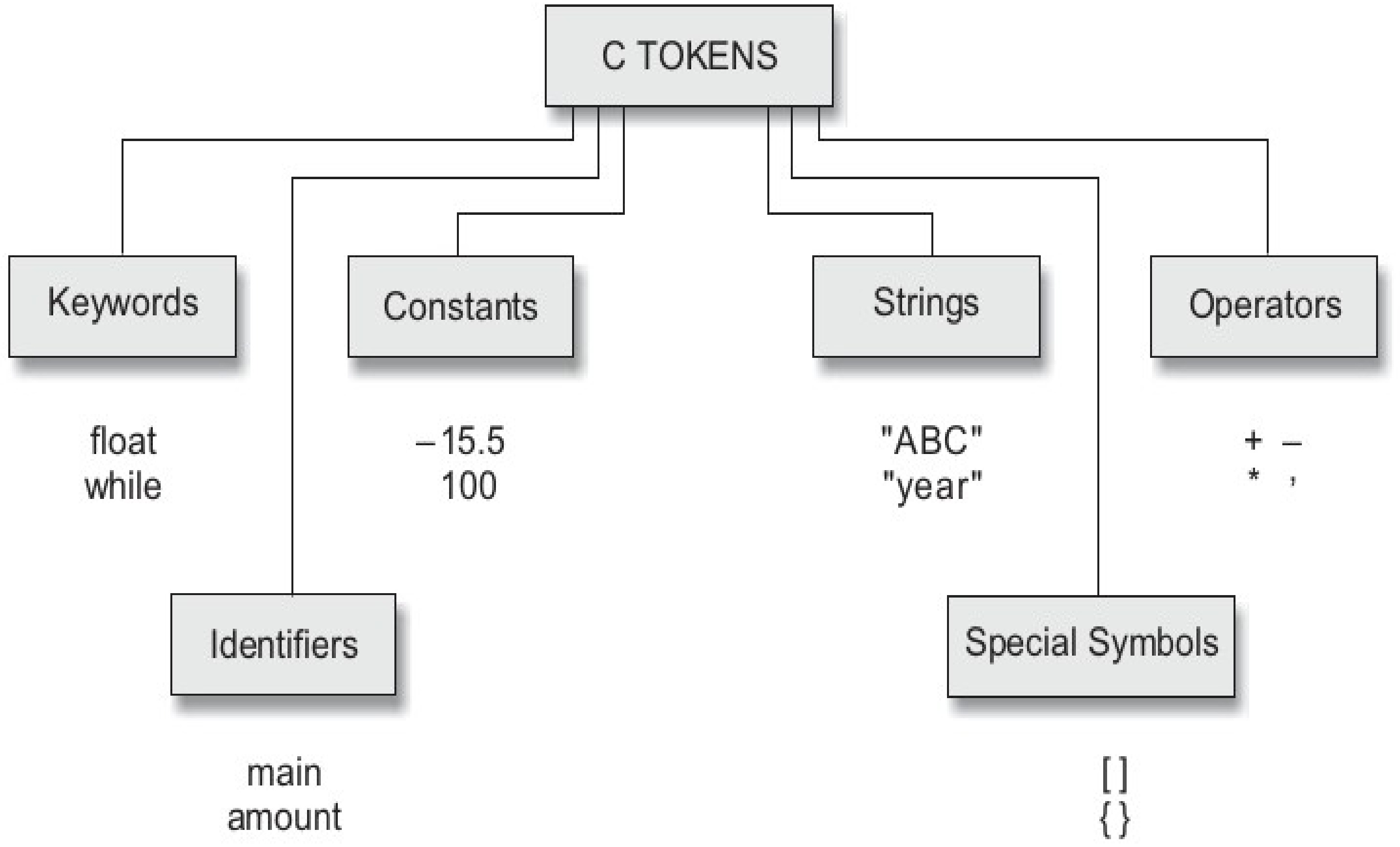
  therefore don't worry */

- The compiler changes each comment into a single blank character. Thus the comments are not part of the executable program.
- Latter single line comment was introduced after c++ came into existence and C also started supporting single line comment which starts with // . Example:
- // this is line 1

  // this is line 2

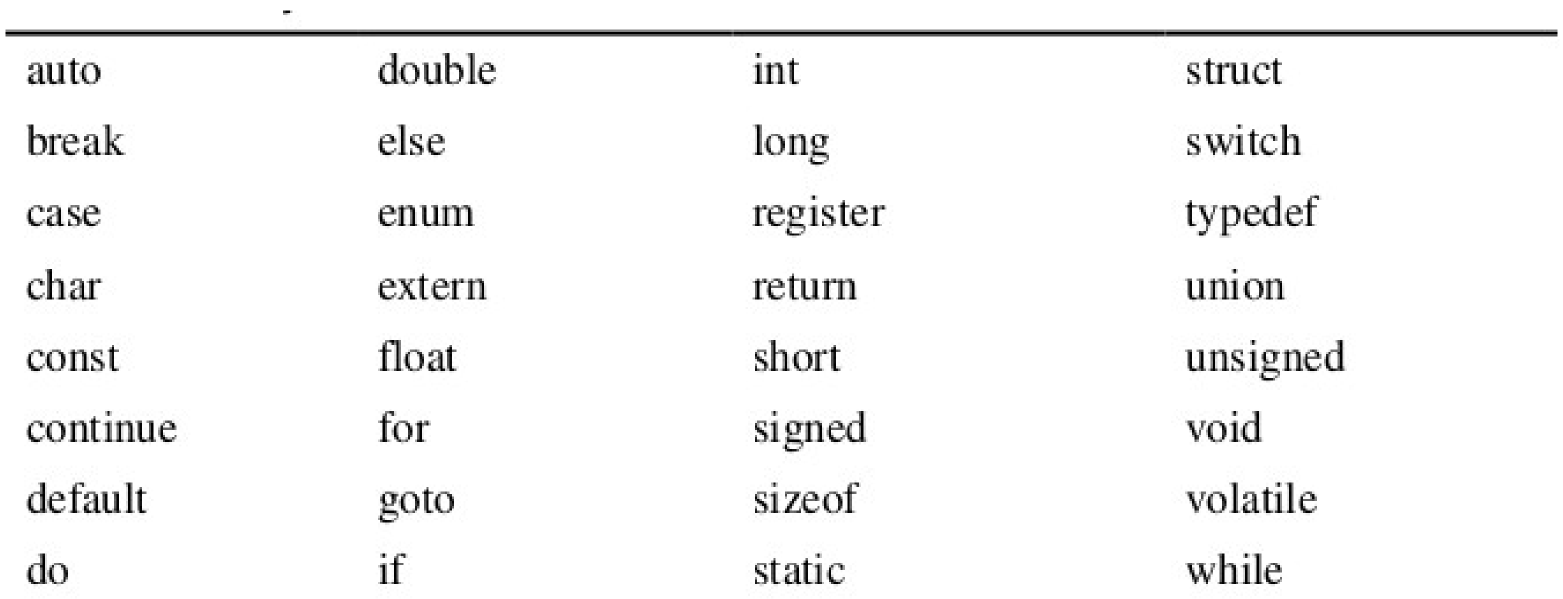
C Tokens

- ✓ **In C program, the smallest individual units known to compiler are known as <u>tokens</u>.**



C Tokens: Keywords

- ✓ The C keywords are reserved words by the compiler. All the C keywords have been assigned fixed meanings and they cannot be used as variable names.
- ✓ Every C word is classified as either a keyword or an identifier.
- ✓ List of 32 keywords defined by ANSI Standard are given below:



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

C Tokens: Identifiers

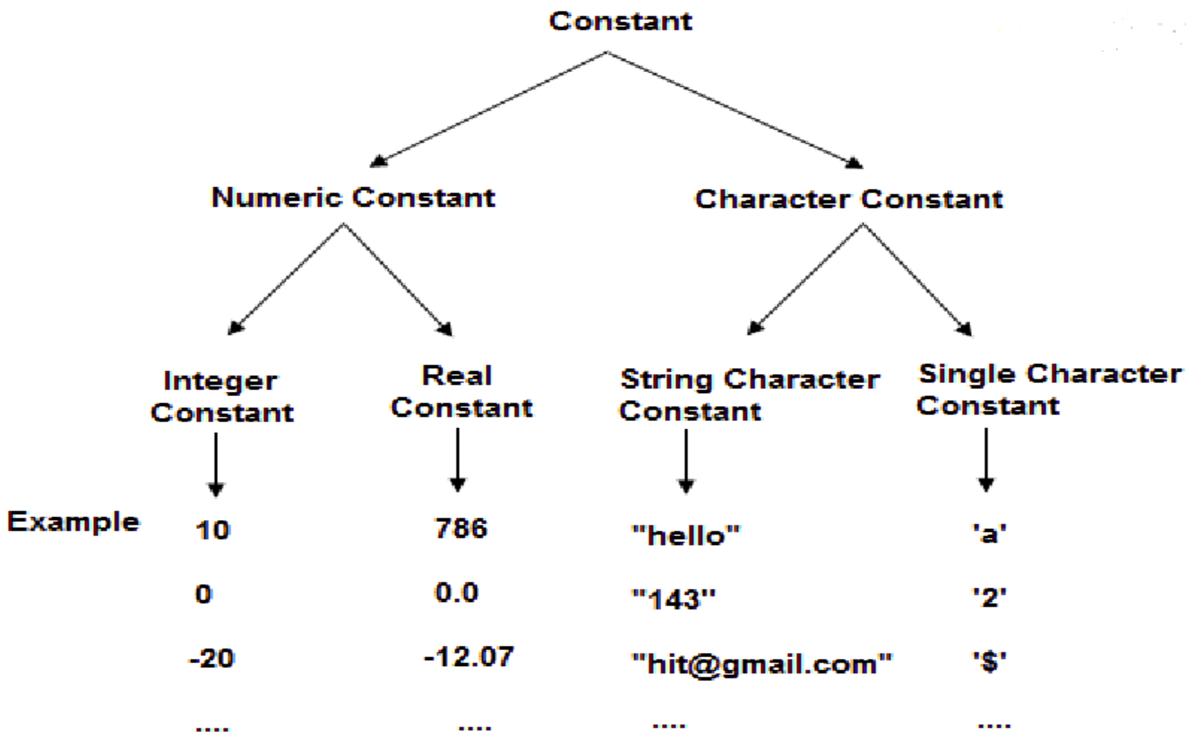
A symbolic name is used to refer to a variable, constant, function, structure, union etc. Identifiers refer to the names of variables, functions, arrays, structure, labels ,etc to identify them.

Rules for naming an identifier in C

- An identifier can only have alphanumeric characters (a-z , A-Z , 0-9) (i.e. letters &
- digits) and underscore(_) symbol.
- Identifier names must be unique.
- The first character must be an alphabet or underscore.
- You cannot use a keyword as identifiers.
- Only the first thirty-one (31) characters are significant. [ANSI C]
- It must not contain white spaces.
- Identifiers are case-sensitive.

C Tokens: Constants

- ✓ Constants in C refer to fixed values created by programmer in the current program that do not change during the execution of a program. Basic Types of C constants are shown below:



C Tokens: Strings

Strings in C is an array of characters having null character ‘\0’ at the end. Strings in C are enclosed in double-quotes(“ ”) and Characters are enclosed in single quotes(‘ ’).

<u>Example of strings:</u>

```
char name[20]= “apple”;
char name2[20]= {‘a’,’p’,’p’,’l’,’e’,’\0’};
char word[20] = “9860”;
```

C Tokens: Special Symbols

These symbols are different from operators and hold special purpose.

Square brackets [ ]: Used for single and multi-dimensional arrays.

Simple brackets ( ): Used for function declaration.

Curly braces { }: Used for opening and closing the code.

The comma (,): Used to separate variables as well as operator.

Hash/pre-processor (#): Used for the header file.

Asterisk (*): Used for Pointers.

Tilde (~): Used for destructing the memory.

Period (.): Used for accessing union and structure members.

C Tokens: Operators

- ✓ This is used to perform special operations on data/operand.
- ✓ Unary Operator: Applied with a single operand. Eg : a++;
- ✓ Binary Operator: Applied between 2 operands. Eg a+b;

Arithmetic Operators

Relational Operators

Shift Operators

Logical Operators

Bitwise Operators

Conditional Operators

Assignment Operator

Misc Operator

Escape Sequence

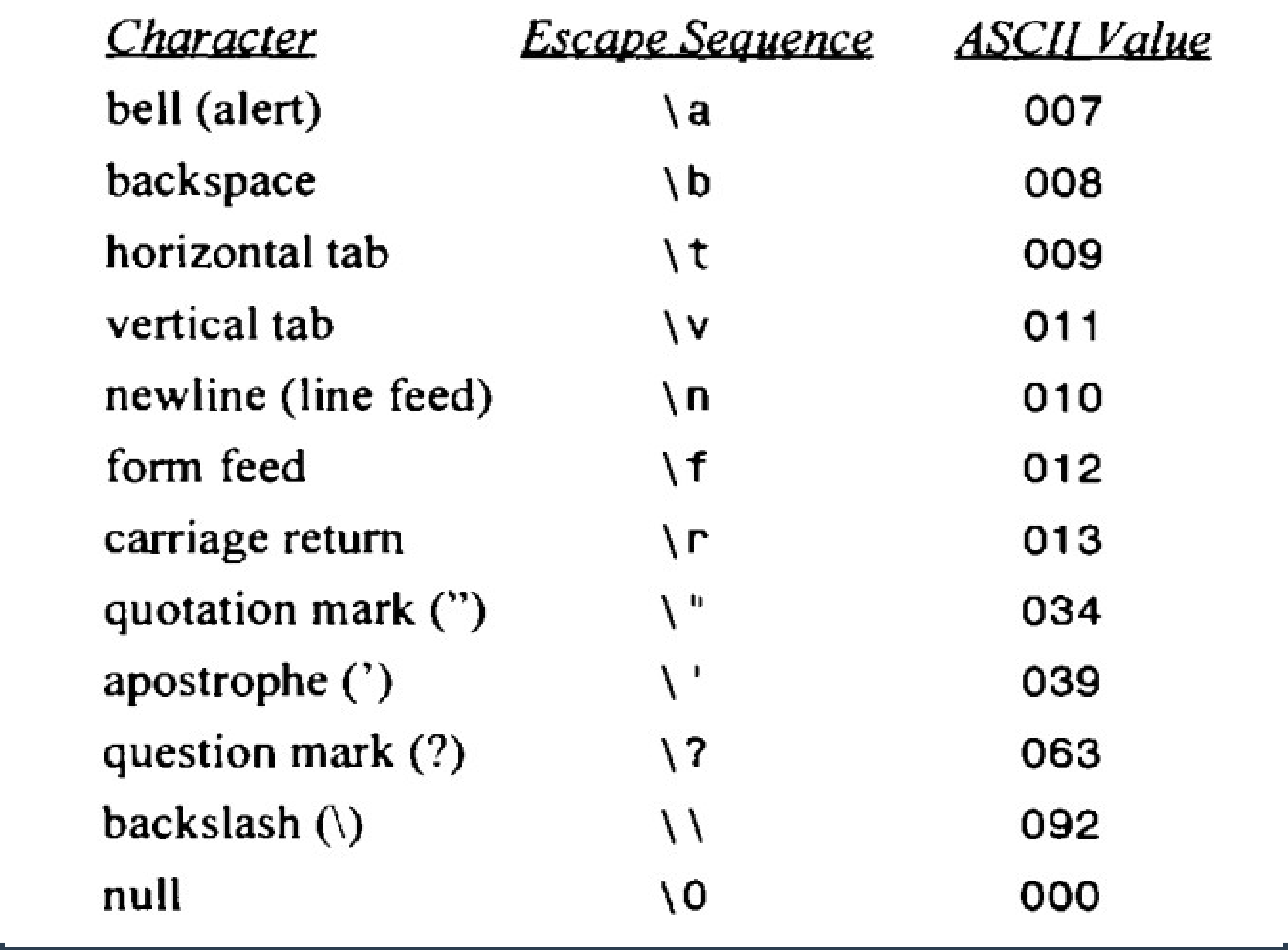
- ✓ Escape sequences are one of the most striking and useful features of the C programming language. Escape sequences provide different meanings to your program code.

- ✓ From the name, you can easily state that in an escape sequence, a character denotes a different meaning from its actual meaning and undergoes a change from its normal form. Each Escape sequence is denoted by a backslash('\').

- ✓ It is a sequence of characters which is used for formatting the output in a structured and beautiful way. But it did not get displayed on the screen while printing the output.

Escape Sequence

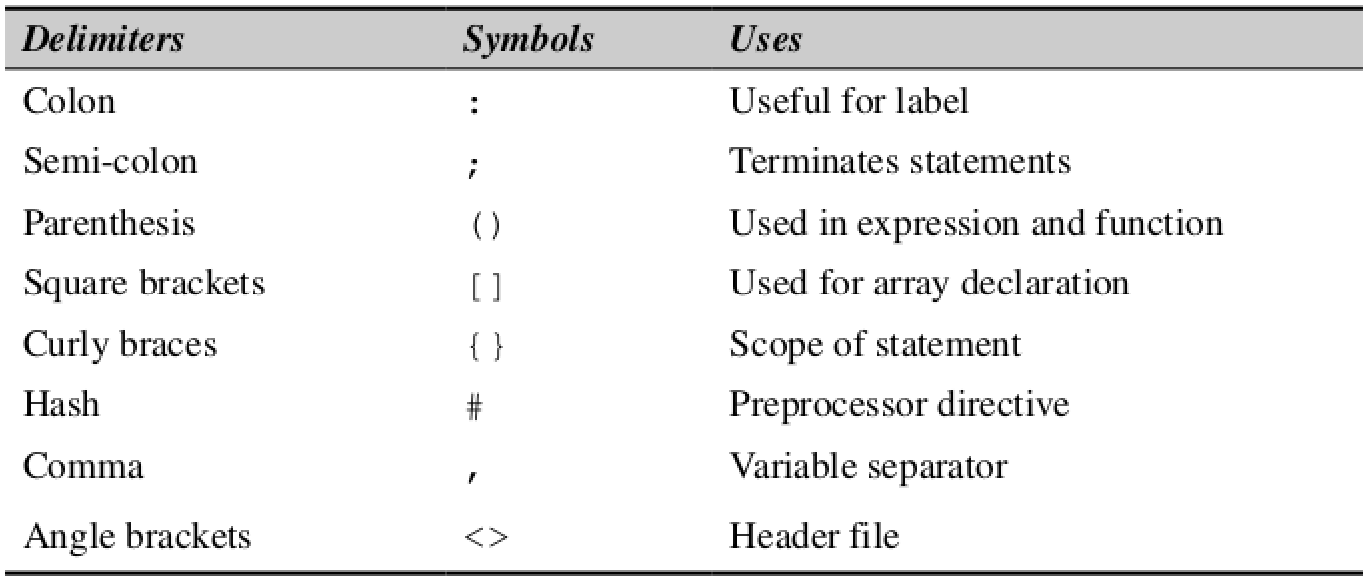
- ✓ List of escape sequences in C:



Character	*Escape Sequence*	*ASCII Value*
bell (alert)	\a	007
backspace	\b	008
horizontal tab	\t	009
vertical tab	\v	011
newline (line feed)	\n	010
form feed	\f	012
carriage return	\r	013
quotation mark (”)	\"	034
apostrophe (’)	\'	039
question mark (?)	\?	063
backslash (\)	\\	092
null	\0	000

Delimiters

- ✓ A delimiter is a symbol that is used to separate (identify the beginning or the end of) the tokens such as identifiers, keywords, constants in the statement. They are also called as separators sometimes.



Delimiters	*Symbols*	*Uses*
Colon	:	Useful for label
Semi-colon	;	Terminates statements
Parenthesis	( )	Used in expression and function
Square brackets	[ ]	Used for array declaration
Curly braces	{ }	Scope of statement
Hash	#	Preprocessor directive
Comma	,	Variable separator
Angle brackets	<>	Header file

Variables

- ✓ A variable is a name given to a memory block/storage area used for storing a data value.
- ✓ A variable is a data name used for storing a data value. Its value may be changed during the program execution. In other words, a variable can be assigned different values at different times during the program execution.
- ✓ A variable must be declared only once, before using it. Declaration gives an introduction of variable to compiler and its properties like scope, range of values and memory required for storage. It has memory location and can store single value at a time.
- ✓ Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.
- ✓ Example:

```
int a ; // variable a is declared //declaration
a=15;  // initialization // using the declared variable
```

Constants/Literals

- ✓ The term constant or literals refer to the fixed value which is never changed during the execution of a program. The following types of literals are available in C:

1. Integer-constant
2. Character-constant
3. Floating-constant
4. String-constant

Example:

printf("%d", 2123); // integer constant

- ✓ A symbolic constant is a name given to some numeric constant, or a character constant or string constant, or any other constants. Symbolic constant names are also known as constant identifiers. Pre-processor directive #define is used for defining symbolic constants.

Example:

#define PI 3.15   // floating point symbolic constant

Data Types

- ✓ C supports several different types of data, each of which may be represented differently within the computer’s memory. A data type can be defined as an abstract concept that defines an internal representation of data in the memory.

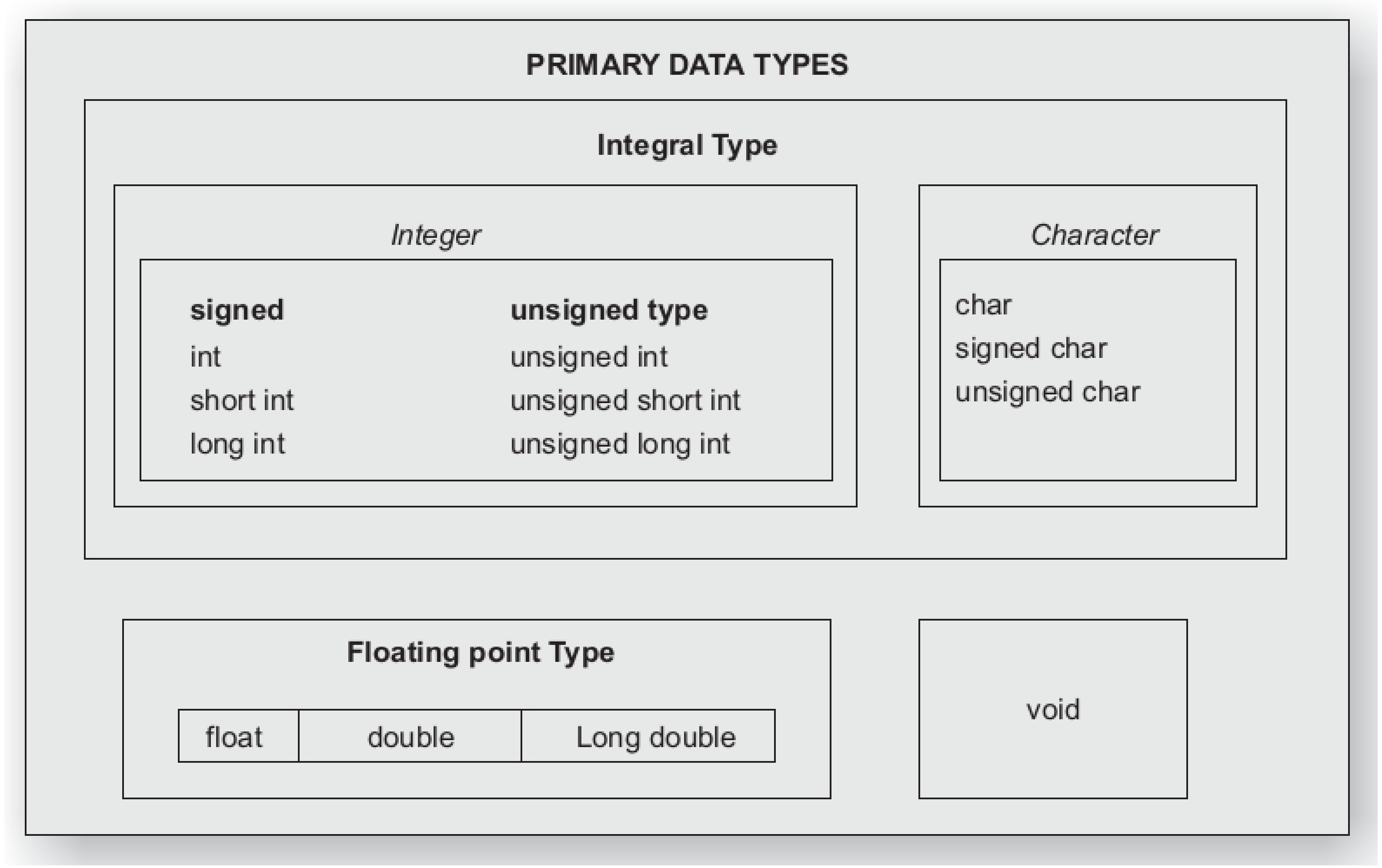
- ✓ Data types define the type of data a variable can hold. For example an integer variable can hold integer data, a character type variable can hold character data.

- ✓ C language is rich in its data types. The variety of data types available allow the programmer to select the type appropriate to the needs of the application as well as the machine. ANSI C supports three classes of data types:

1. Primary (or fundamental) data types
2. Derived data types
3. User-defined data types

Data Types: Primary Data Type

- Those data-types that are pre-defined in C and does not require the help of any other data type because they already exists in C.



Data Types: Derived Data type

- ✓ These are those data types which need the help of derived data type. These data type are not defined in C.
- ✓ Array, string, function, pointer.

Data Types: User-defined Datatype

- ✓ Those data types which are defined by the user as per his/her will are called user-defined data types. Example : enumeration, structure, union.

typedef

C supports a feature known as “type definition” that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables.

The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

- ✓ Example:

  typedef int scale;
- ✓ scale length, breadth;

Expressions

The expression may consist of a single entity, such as a constant or variable, or it may consist of some combination of such entities, interconnected by one or more operators. Expressions can also represent logical conditions which are either true or false. Several simple expressions are given below:

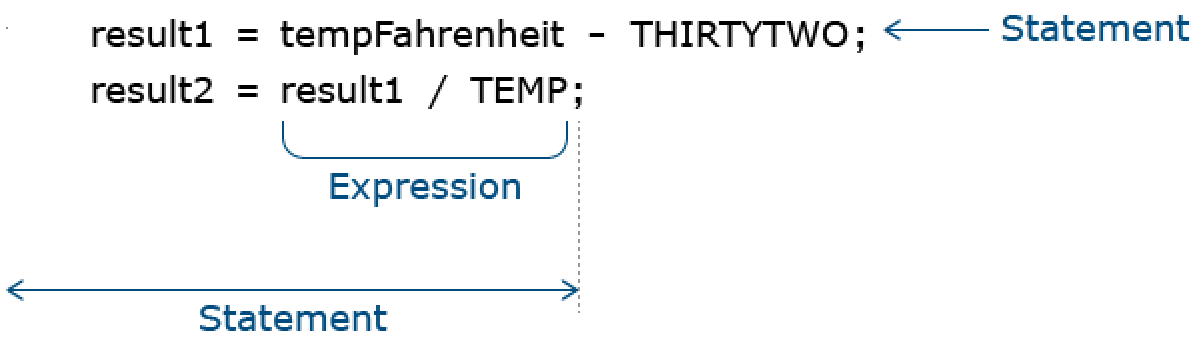
a + b

x = y

t = u + v

x <= y

++j



```
result1 = tempFahrenheit - THIRTYTWO;
result2 = result1 / TEMP;
```

Statements

An expression such as x = 0 or i++ or printf(...) becomes a statement when it is followed by a semicolon, as in

```
x = 0;
i++;
printf(...);
```

A statement causes the computer to carry out some definite action. There are three different classes of statements in C: expression statements, compound statements, and control statements.

An expression statement consists of an expression followed by a semicolon. The execution of such a statement causes the associated expression to be evaluated. For example:

```
a = 6;
```

A compound statement consists of several individual statements enclosed within a pair of braces { }. A typical compound statement is shown below:

```
{
 pi = 3.141593;
 circumference = 2. * pi * radius;
}
```

A control statements are those statements which causes the change in the sequential execution of the program. Example if..else

Library Functions

- ✓ Library functions are a collection of inbuilt functions grouped together and stored in a Library file on the disk. This library of functions comes ready-made (already defined) in C.
- ✓ Library Functions are declared in the C header files such as scanf(), printf() are defined in stdio.h header file, similarly, pow() and sqrt() function are defined in math.h header file.

Example:

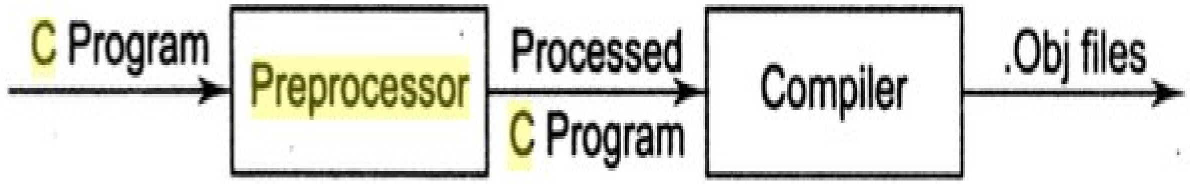
```
#include<math.h>

int a=4;

sqrt(a); // using sqrt() library function to calculate the square root of ‘a’
```

Preprocessor Directives.

- ✓The C preprocessor, as its name indicates, process the C program before it is compiled.
- ✓The preprocessor transforms the program as specified by the preprocessor instructions given in the program.
- ✓These instructions, which are different from C instructions , begins with # character known as preprocessor directives.
- ✓The C preprocessor is automatically invoked by the C compiler before a program is compiled.
- ✓C processor is not the part of compiler.



Preprocessor Directives.

Preprocessor directives can be divided into following category:

(a) Macro expansion / Macro substitution directives

(b) File inclusion directive

(c) Conditional compilation directive

(d) Miscellaneous directives

Preprocessor Directives.

a) Macro Expansion : Macro is one of the categories of a preprocessor directive.A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive. During preprocessing, every macro template gets replaced with its corresponding macro expansion (substitute value).

There are different forms of macro substitution. The most common forms are:

1. Simple macro substitution.

```
#include <stdio.h>

#define PI 3.14

int main() {

   float area,radius=4.35;

   area = PI*radius*radius;

   printf("Area is %f", area);

   return 0;

}
```

Preprocessor Directives.

2. Argumented macro substitution.

```
#include <stdio.h>

#define SQUARE(x) x*x

int main() {

    float result;

    result = SQUARE(6);

    printf("Square is %f", result);

    return 0;

}
```

Preprocessor Directives.

3. Nested macro substitution (parameterized).

```
#include <stdio.h>

#define SQUARE(x) x*x

#define CUBE(x) (SQUARE(x) * x)

int main() {

    float result;

    result = CUBE(6);

    printf("CUBE is %f", result);

    return 0;

}
```

Q. Find the area of circle using simple, argumented and nested macro .

Preprocessor Directives.

(b) File inclusion directive : We can use file inclusion directive through #include and followed by the file-name. It causes the entire contents of filename to be inserted into the source code where we have used #include.

Example:

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

Q. Include your own header file

Preprocessor Directives.

(c) Conditional compilation directive:

Conditional compilation : Using special preprocessing directives, you can include or exclude parts of the program according to various conditions. Eg:

```
#ifndef MESSAGE    //#ifdef macro

      #define MESSAGE "You wish!"

#endif

-----------------------------------------------------------------------------------------

#include<stdio.h>

#define MESSAGE 1

#if MESSAGE==1   // #if expression

      #define MESSAGE "You wish!"

#endif
```

Preprocessor Directives.

(d) Miscellaneous directives

#undef

#pragma

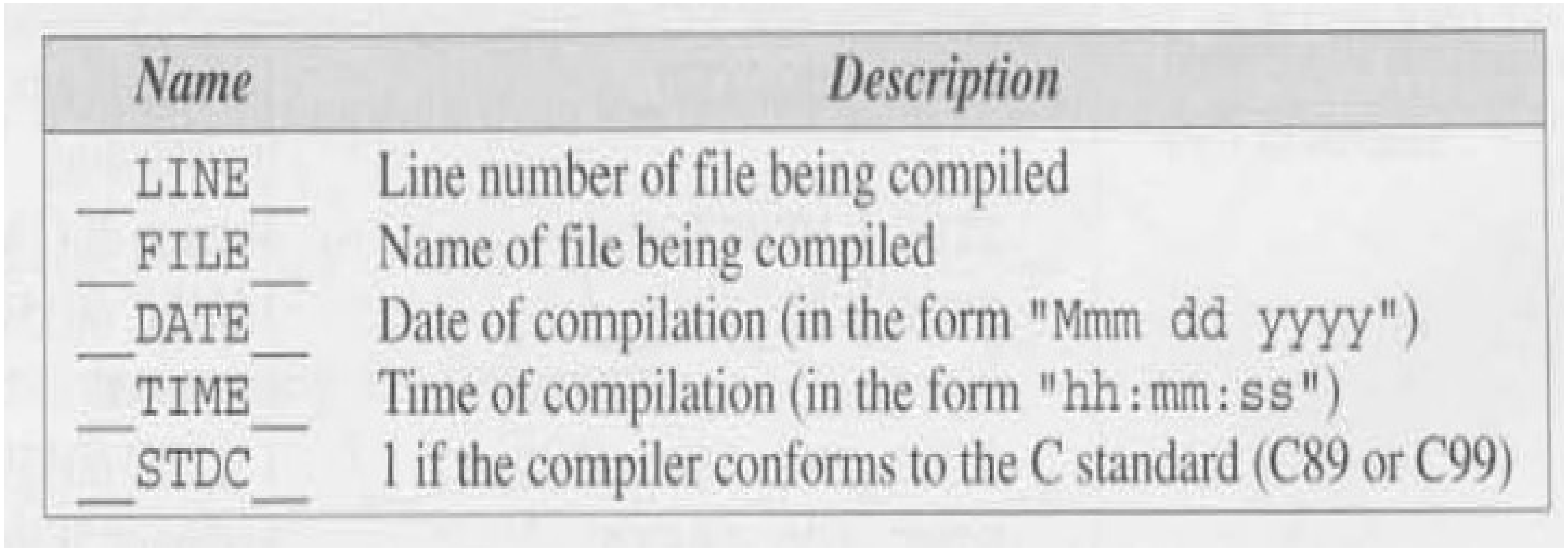
#pragma startup function_name 105 // 105 is priority number; lower is higher

#pragma exit function_name

#line	__LINE__
#error	#error message

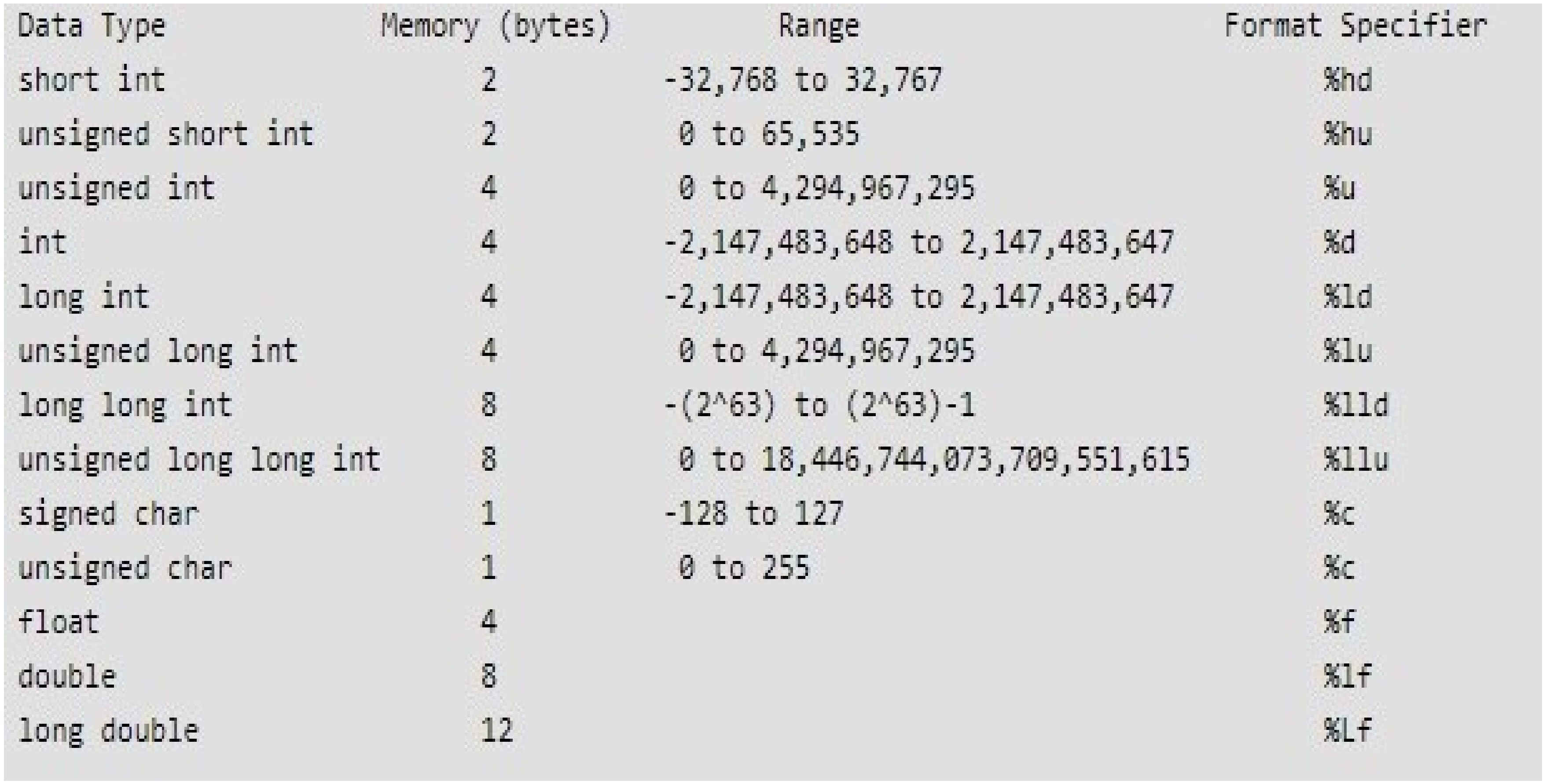
Preprocessor Directives.

Predefined Macros in C



Name	*Description*
__LINE__	Line number of file being compiled
__FILE__	Name of file being compiled
__DATE__	Date of compilation (in the form "Mmm dd yyyy")
__TIME__	Time of compilation (in the form "hh:mm:ss")
__STDC__	1 if the compiler conforms to the C standard (C89 or C99)

Data type and Conversion Specifier



Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	12		%Lf