PROBLEM SOLVING TECHNIQUES USING C

Total Teaching Hours: 52 No of Hours / Week: 04

Unit – I 12 Hours

Introduction to Programming Concepts: Software, Classification of Software, Modular Programming, Structured Programming, Algorithms and Flowcharts with examples. Overview of C Language: History of C, Character set, C tokens, Identifiers, Keywords, Data types, Variables, Constants, Symbolic Constants, Operators in C, Hierarchy of Operators, Expressions, Type Conversions and Library Functions.

Unit - II 10 Hours

Managing Input and Output Operation: Formatted and Unformatted I/O Functions, Decision making, branching and looping: Decision Making Statements - if Statement, if–else statement, nesting of if-else statements, else–if ladder, switch statement, operator, Looping - while, do-while, for loop, Nested loop, break, continue, and go-to statements. Functions: Function Definition, prototyping, types of functions, passing arguments to functions, Nested Functions, Recursive functions.

Unit - III 10 Hours

Arrays: Declaring and Initializing, One Dimensional Arrays, Two Dimensional Arrays, Multi-Dimensional Arrays - Passing arrays to functions. Strings: Declaring and Initializing strings, Operations on strings, Arrays of strings, passing strings to functions. Storage Classes - Automatic, External, Static and Register Variables.

Unit-IV 10 Hours

Structures-Declaring and Initializing, Nested structure, Array of Structure, Passing Structures to functions, Unions, typedef, enum, Bit fields. Pointers – Declarations, Pointer arithmetic, Pointers and functions, call by value, Call by reference, Pointers and Arrays, Arrays of Pointers, Pointers and Structures. Meaning of static and dynamic memory allocation, Memory allocation functions. 5

Unit-V 10 Hours

Files - File modes, File functions, and File operations, Text and Binary files, Command Line arguments. C Pre-processor directives, Macros – Definition, types of Macros, Creating and implementing user defined header files.

TEXT BOOKS

1. E. Balaguruswamy, "Programming In ANSI C", 4th edition, TMH Publications, 2007

2. Manjunath Aradhya, " Programming and Data Structures" , Cengage Publications 2017

3. A. K. Sharma, "Computer Fundamentals and Programming in C", University Press, 2018

REFERENCES BOOKS

1. Ashok N. Kamthane et. al., "Computer Programming and IT", Pearson Education, 2011

2. Mahapatra, " Thinking In C ", PHI Publications, 1998.

3. Yashwant Kanetkar, "Let Us C", 13th Edition, PHP, 2013

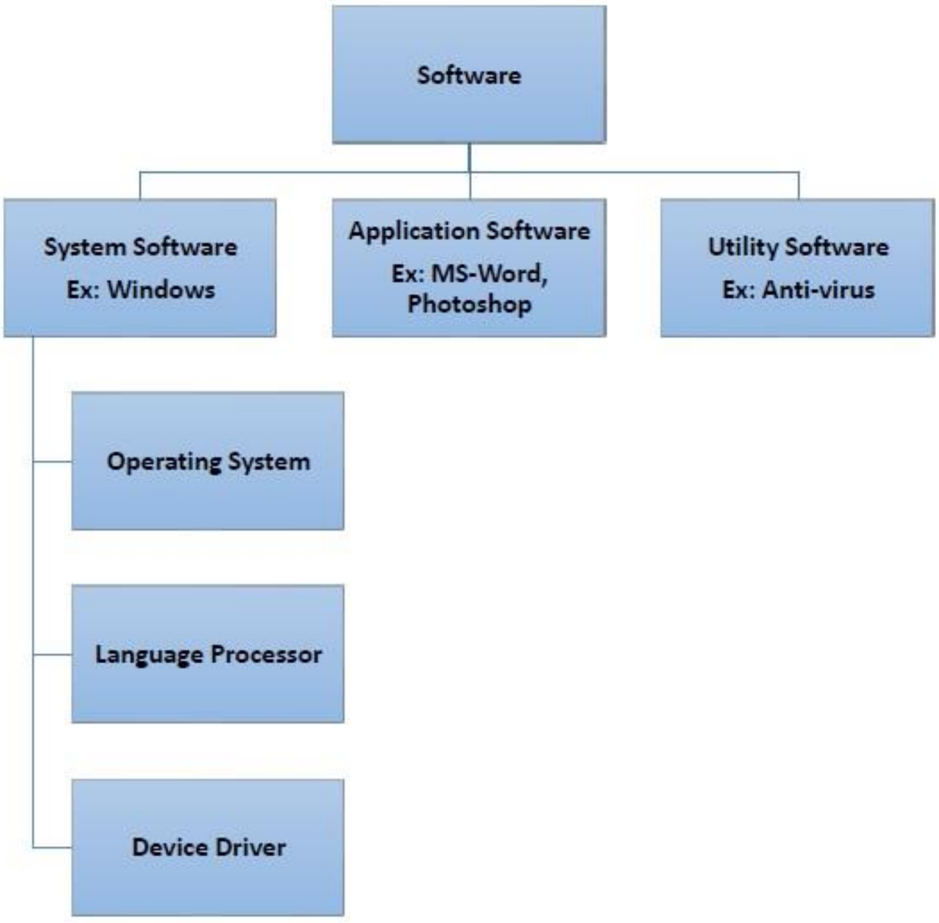
Introduction to Programming Concepts: Software, Classification of Software

As you know, the hardware devices need user instructions to function. A set of instructions that achieve a single outcome are called program or procedure. Many programs functioning together to do a task make a **software**.

For example, a word-processing software enables the user to create, edit and save documents. A web browser enables the user to view and share web pages and multimedia files. There are two categories of software −

- System Software
- Application Software
- Utility Software

Let us discuss them in detail.



System Software

Software required to run the hardware parts of the computer and other application software are called **system software**. System software acts as **interface** between hardware and user applications. An interface is needed because hardware devices or machines and humans speak in different languages.

Machines understand only binary language i.e. 0 (absence of electric signal) and 1 (presence of electric signal) while humans speak in English, French, German, Tamil, Hindi and many other languages. English is the pre-dominant language of interacting with computers. Software is required to convert all human instructions into machine understandable instructions. And this is exactly what system software does.

Based on its function, system software is of four types −

- Operating System
- Language Processor
- Device Drivers

Operating System

System software that is responsible for functioning of all hardware parts and their interoperability to carry out tasks successfully is called **operating system (OS)**. OS is the first software to be loaded into computer memory when the computer is switched on and this is called **booting**. OS manages a computer's basic functions like storing data in memory, retrieving files from storage devices, scheduling tasks based on priority, etc.

Language Processor

As discussed earlier, an important function of system software is to convert all user instructions into machine understandable language. When we talk of human machine interactions, languages are of three types −

- **Machine-level language** − This language is nothing but a string of 0s and 1s that the machines can understand. It is completely machine dependent.
- **Assembly-level language** − This language introduces a layer of abstraction by defining **mnemonics**. **Mnemonics** are English like words or symbols used to denote a long string of 0s and 1s. For example, the word "READ" can be defined to mean that computer has to retrieve data from the memory. The complete **instruction** will also tell the memory address. Assembly level language is **machine dependent**.
- **High level language** − This language uses English like statements and is completely independent of machines. Programs written using high level languages are easy to create, read and understand.

Program written in high level programming languages like Java, C++, etc. is called **source code**. Set of instructions in machine readable form is called **object code** or **machine code**. **System software** that converts source code to object code is called **language processor**. There are three types of language interpreters−

- **Assembler** − Converts assembly level program into machine level program.
- **Interpreter** − Converts high level programs into machine level program line by line.
- **Compiler** − Converts high level programs into machine level programs at one go rather than line by line.

Device Drivers

System software that controls and monitors functioning of a specific device on computer is called **device driver**. Each device like printer, scanner, microphone, speaker, etc. that needs to be attached externally to the system has a specific driver associated with it. When you attach a new device, you need to install its driver so that the OS knows how it needs to be managed.

Application Software

A software that performs a single task and nothing else is called **application software**. Application software are very specialized in their function and approach to solving a problem. So a spreadsheet software can only do operations with numbers and nothing else. A hospital management software will manage hospital activities and nothing else. Here are some commonly used application software −

- Word processing
- Spreadsheet
- Presentation
- Database management
- Multimedia tools

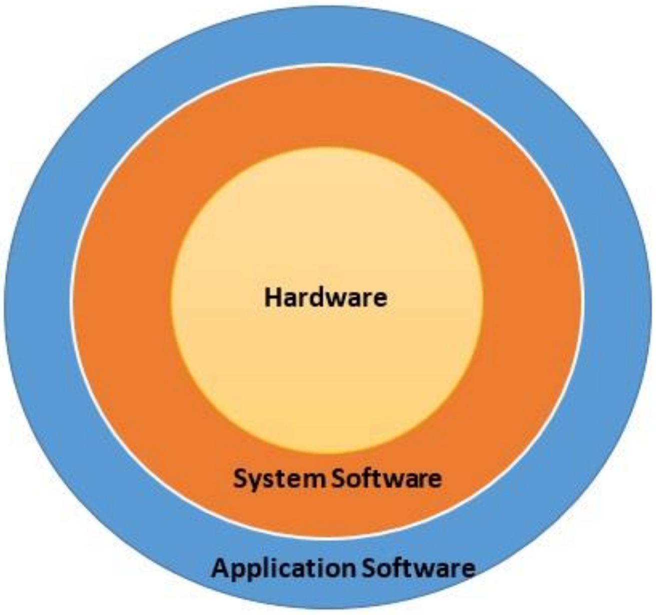
Utility Software

Application software that assist system software in doing their work is called **utility software**. Thus utility software is actually a cross between system software and application software. Examples of utility software include −

- Antivirus software
- Disk management tools
- File management tools

- Compression tools
- Backup tools

As you know, system software acts as an interface for the underlying hardware system. Here we will discuss some important system software in detail.



Operating System

Operating system (OS) is the lifeline of computer. You connect all the basic devices like CPU, monitor, keyboard and mouse; plug in the power supply and switch it on thinking you have everything in place. But the computer will not start or come to life unless it has an operating system installed in it because OS −

- Keeps all hardware parts in a state of readiness to follow user instructions
- Co-ordinates between different devices
- Schedules multiple tasks as per priority
- Allocates resource to each task
- Enables computer to access network

- Enables users to access and use application software

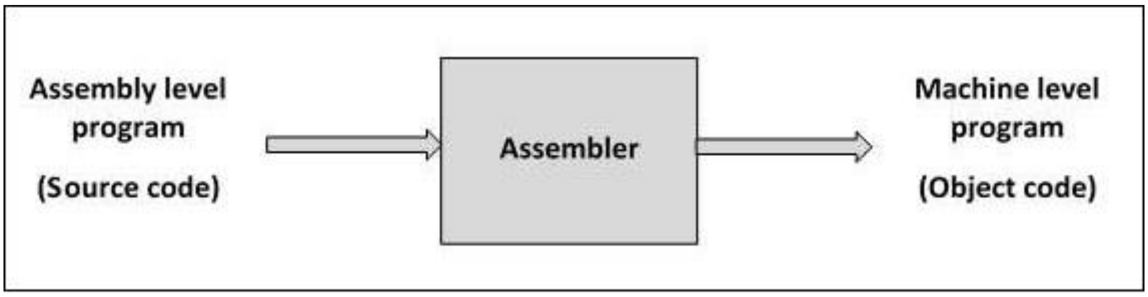
Besides initial booting, these are some of the functions of an operating system −

- Managing computer resources like hardware, software, shared resources, etc.
- Allocating resources
- Prevent error during software use
- Control improper use of computer

One of the earliest operating systems was **MS-DOS,** developed by Microsoft for IBM PC. It was a **Command Line Interface (CLI)** OS that revolutionized the PC market. DOS was difficult to use because of its interface. The users needed to remember instructions to do their tasks. To make computers more accessible and user-friendly, Microsoft developed **Graphical User Interface (GUI)** based OS called **Windows**, which transformed the way people used computers.

Assembler

Assembler is a system software that converts assembly level programs to machine level code.



These are the advantages provided by assembly level programming −

- Increases efficiency of the programmer as remembering mnemonics is easier

- Productivity increases as number of errors decreases and hence debugging time
- Programmer has access to hardware resources and hence has flexibility in writing programs customized to the specific computer

Interpreter

The major advantage of assembly level language was its ability to optimize memory usage and hardware utilization. However, with technological advancements computers had more memory and better hardware components. So ease of writing programs became more important than optimizing memory and other hardware resources.

In addition, a need was felt to take programming out of a handful of trained scientists and computer programmers, so that computers could be used in more areas. This led to development of high level languages that were easy to understand due to resemblance of commands to English language.

The system software used to translate high level language source code into machine level language object code line by line is called an **interpreter**. An interpreter takes each line of code and converts it into machine code and stores it into the object file.

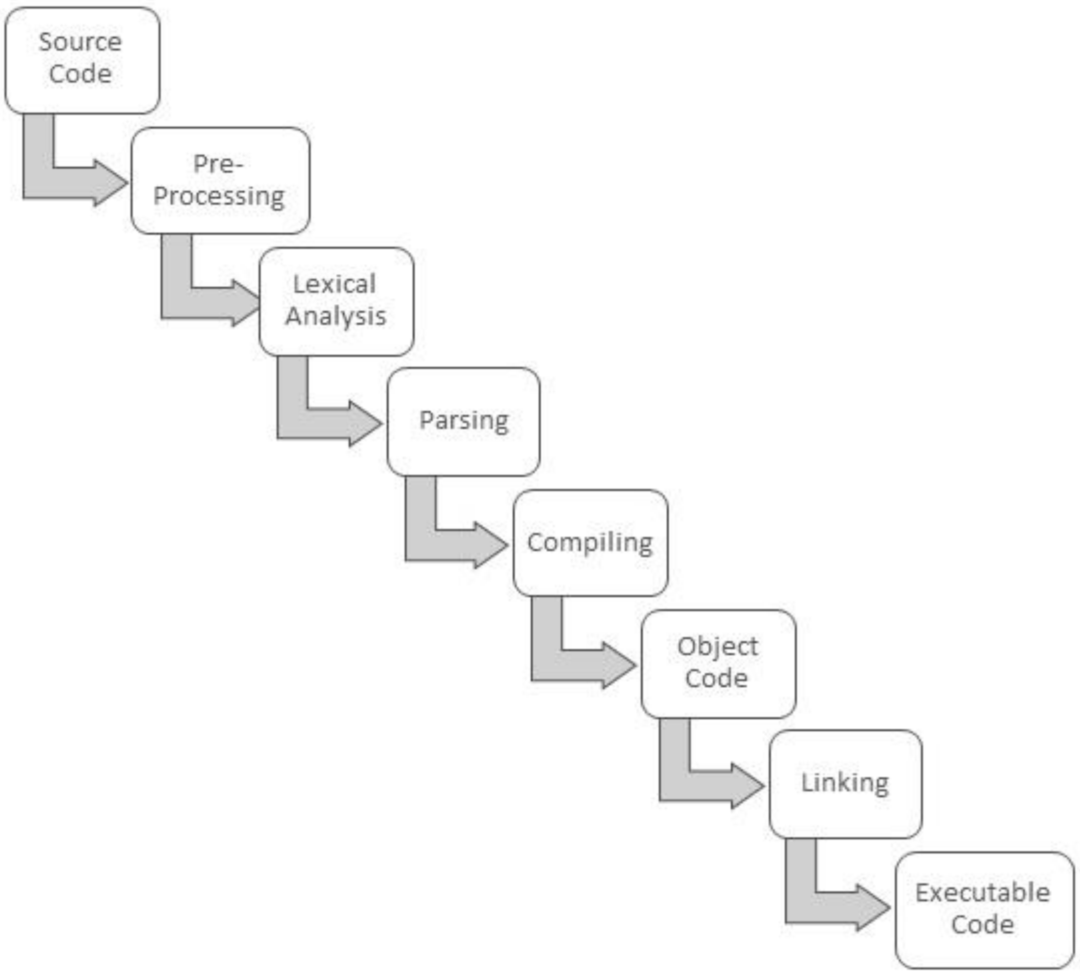
The **advantage** of using an interpreter is that they are very easy to write and they do not require a large memory space. However, there is a major disadvantage in using interpreters, i.e., interpreted programs take a long time in executing. To overcome this **disadvantage**, especially for large programs, **compilers** were developed.

Compiler

System software that store the complete program, scan it, translate the complete program into object code and then creates an executable code is called a compiler. On the face of it compilers compare unfavorably with interpreters because they −

- are more complex than interpreters
- need more memory space
- take more time in compiling source code

However, compiled programs execute very fast on computers. The following image shows the step-by-step process of how a source code is transformed into an executable code −



These are the steps in compiling source code into executable code −

- **Pre-processing** − In this stage pre-processor instructions, typically used by languages like C and C++ are interpreted, i.e. converted to assembly level language.
- **Lexical analysis** − Here all instructions are converted to **lexical units** like constants, variables, arithmetic symbols, etc.
- **Parsing** − Here all instructions are checked to see if they conform to **grammar rules** of the language. If there are errors, compiler will ask you to fix them before you can proceed.
- **Compiling** − At this stage the source code is converted into **object code**.
- **Linking** − If there are any links to external files or libraries, addresses of their executable will be added to the program. Also, if the code needs to be rearranged for actual execution, they will be rearranged. The final output is the **executable code** that is ready to be executed.

Structured Programming Approach with Advantages and Disadvantages

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:

- C

The structured program mainly consists of three types of elements:

- Selection Statements
- Sequence Statements

- Iteration Statements

The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach:

1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

Modular programming is a software design technique that emphasizes separating the functionality of a **program** into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Or

Modular Programming allows development to be divided by splitting down a **program** into smaller programs in order to execute a variety of tasks. This enables developers to work simultaneously and minimizes **the** time taken for development.

Advantages of Using Modular Programming Approach –

1. **Ease to Use :**This approach allows simplicity, as rather than focusing on the entire thousands and millions of lines code in one go we can access it in the form of modules. This allows ease in debugging the code and prone to less error.
2. **Reusability: It** allows the user to reuse the functionality with a different interface without typing the whole program again.
3. **Ease of Maintenance:** It helps in less collision at the time of working on modules, helping a team to work with proper collaboration while working on a large application.

History of C

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons −

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platform

Being a middle-level language, C reduces the gap between the low-level and high-level languages. It can be used for writing operating systems as well as doing application level programming. Helps to understand the fundamentals of Computer Theories.

What is C programming in simple words?

It is a procedural **language**, which means that people can write their **programs** as a series of step-by-step instructions.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

Where is C used? Key Applications

1. 'C' language is widely used in embedded systems.
2. It is used for developing system applications.

3. It is widely used for developing desktop applications.
4. Most of the applications by Adobe are developed using 'C' programming language.
5. It is used for developing browsers and their extensions. Google's Chromium is built using 'C' programming language.
6. It is used to develop databases. MySQL is the most popular database software which is built using 'C'.
7. It is used in developing an operating system. Operating systems such as Apple's OS X, Microsoft's Windows, and Symbian are developed using 'C' language. It is used for developing desktop as well as mobile phone's operating system.
8. It is used for compiler production.
9. It is widely used in IOT applications.

Why learn 'C'?

As we studied earlier, 'C' is a base language for many programming languages. So, learning 'C' as the main language will play an important role while studying other programming languages. It shares the same concepts such as data types, operators, control statements and many more. 'C' can be used widely in various applications. It is a simple language and provides faster execution. There are many jobs available for a 'C' developer in the current market.

'C' is a structured programming language in which program is divided into various modules. Each module can be written separately and together it forms a single 'C' program. This structure makes it easy for testing, maintaining and debugging processes.

'C' contains 32 keywords, various data types and a set of powerful built-in functions that make programming very efficient.

Another feature of 'C' programming is that it can extend itself. A 'C' program contains various functions which are part of a library. We can add our features and functions to the library. We can access and use these functions anytime we want in our program. This feature makes it simple while working with complex programming.

Various compilers are available in the market that can be used for executing programs written in this language.

It is a highly portable language which means programs written in 'C' language can run on other machines. This feature is essential if we wish to use or execute the code on another computer.

Summary

- 'C' was developed by Dennis Ritchie in 1972.
- It is a robust language.
- It is a low programming level language close to machine language
- It is widely used in the software development field.
- It is a procedure and structure oriented language.
- It has the full support of various operating systems and hardware platforms.
- Many compilers are available for executing programs written in 'C'.
- A compiler compiles the source file and generates an object file.
- A linker links all the object files together and creates one executable file.
- It is highly portable.

Character set and C tokens

Like every other language, 'C' also has its own character set. A program is a set of instructions that, when executed, generate an output. The data that is processed by a program consists of various characters and symbols. The output generated is also a combination of characters and symbols.

A character set in 'C' is divided into,

- Letters
- Numbers
- Special characters
- White spaces (blank spaces)

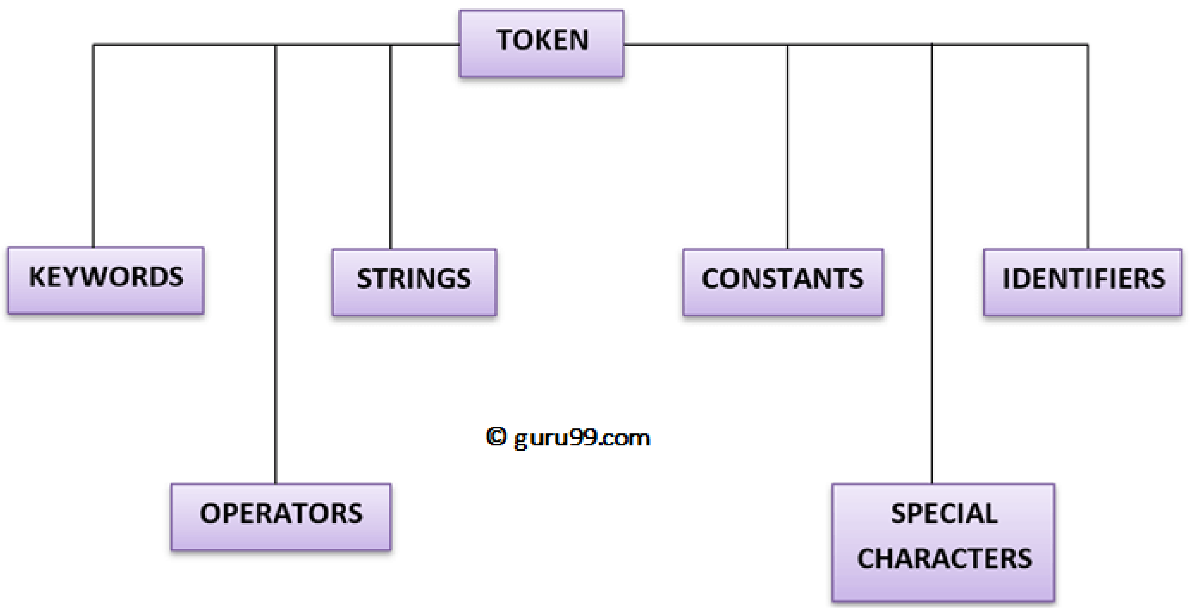
A compiler always ignores the use of characters, but it is widely used for formatting the data. Following is the character set in 'C' programming:

1. Letters
    - Uppercase characters (A-Z)
    - Lowercase characters (a-z)
2. Numbers
    - All the digits from 0 to 9
3. White spaces
    - Blank space
    - New line
    - Carriage return
    - Horizontal tab
4. Special characters
    - Special characters in 'C' are shown in the given table

, (comma)	{ (opening curly bracket)
. (period)	} (closing curly bracket)
; (semi-colon)	[(left bracket)
: (colon)	] (right bracket)
? (question mark)	((opening left parenthesis)
' (apostrophe)	) (closing right parenthesis)
" (double quotation mark)	& (ampersand)
! (exclamation mark)	^ (caret)
|(vertical bar)	+ (addition)
/ (forward slash)	- (subtraction)
\ (backward slash)	* (multiplication)
~ (tilde)	/ (division)
_ (underscore)	> (greater than or closing angle bracket)
$ (dollar sign)	< (less than or opening angle bracket)
% (percentage sign)	# (hash sign)

What is Token in C?

TOKEN is the smallest unit in a 'C' program. It is each and every word and punctuation that you come across in your C program. The compiler breaks a program into the smallest possible units (tokens) and proceeds to the various stages of the compilation. A token is divided into six different types, viz, Keywords, Operators, Strings, Constants, Special Characters, and Identifiers.



Keywords and Identifiers

In 'C' every word can be either a keyword or an identifier.

Keywords have fixed meanings, and the meaning cannot be changed. They act as a building block of a 'C' program. There are a total of 32 keywords in 'C'. Keywords are written in **lowercase** letters.

Following table represents the keywords in 'C'-

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

An identifier is nothing but a name assigned to an element in a program. Example, name of a variable, function, etc. Identifiers are the user-defined names consisting of 'C' standard character set. As the name says, identifiers are used to identify a particular element in a program. Each identifier must have a unique name. Following rules must be followed for identifiers:

1. The first character must always be an alphabet or an underscore.
2. It should be formed using only letters, numbers, or underscore.
3. A keyword cannot be used as an identifier.
4. It should not contain any whitespace character.
5. The name must be meaningful.

Examples of an Identifier.

- Sum or _sum
- Cannot have any special characters except underscore. Ex : sum@ or %difference
- Ex: sum diff plus
- Sum_diff

Summary

- A token is the smallest unit in a program.
- A keyword is reserved words by language.
- There are total of 32 keywords.
- An identifier is used to identify elements of a program.
-

Variables and Constants

What are the variables and constants?

A constant is a data item whose **value** cannot change during the program's execution. Thus, as its name implies – the **value** is constant. A variable is a data item whose **value** can change during the program's execution. Thus, as its name implies – the **value** can vary.

What is a Variable?

A variable is an identifier which is used to store some value. Constants can never change at the time of execution. Variables can change during the execution of a program and update the value stored inside it.

int a,b;

A single variable can be used at multiple locations in a program. A variable name must be meaningful. It should represent the purpose of the variable.

```
Example: Height, age, are the meaningful variables that represent the purpose it i
s being used for. Height variable can be used to store a height value. Age variabl
e can be used to store the age of a person
```

A variable must be declared first before it is used somewhere inside the program. A variable name is formed using characters, digits and an underscore.

Following are the rules that must be followed while creating a variable:

1. A variable name should consist of only characters, digits and an underscore.
2. A variable name should not begin with a number.
3. A variable name should not consist of whitespace.
4. A variable name should not consist of a keyword.
5. 'C' is a case sensitive language that means a variable named 'age' and 'AGE' are different.

Following are the examples of valid variable names in a 'C' program:

```
height or HEIGHT
_height
_height1
My_name
```

Following are the examples of invalid variable names in a 'C' program:

```
1height
Hei$ght
My name
```

For example, we declare an integer variable **my_variable** and assign it the value 48:

```
int my_variable;
my_variable = 48;
```

By the way, we can both declare and initialize (assign an initial value) a variable in a single statement:

```
int my_variable = 48;
```

Constants

Constants are the fixed values that never change during the execution of a program. Following are the various types of constants:

Integer constants

An integer constant is nothing but a value consisting of digits or numbers. These values never change during the execution of a program. Integer constants can be octal, decimal and hexadecimal.

1. Decimal constant contains digits from 0-9 such as,

```
Example, 111, 1234,10,100,
```

Above are the valid decimal constants.

2. Octal constant contains digits from 0-7, and these types of constants are always preceded by 0.

```
Example, 012, 065
```

Above are the valid decimal constants.

3. Hexadecimal constant contains a digit from 0-9 as well as characters from A-F. Hexadecimal constants are always preceded by 0X.

```
Example, 0X2, 0Xbcd
```

Above are the valid hexadecimal constants.

The octal and hexadecimal integer constants are very rarely used in programming with 'C'.

Character constants

A character constant contains only a single character enclosed within a single quote ('   '). We can also represent character constant by providing ASCII value of it.

```
Example, 'A', '9'
```

Above are the examples of valid character constants.

String constants

A string constant contains a sequence of characters enclosed within double quotes ("").

```
Example, "Hello", "Programming", “Addition”, “Multiplication”
```

These are the examples of valid string constants.

Real Constants

Like integer constants that always contains an integer value. 'C' also provides real constants that contain a decimal point or a fraction value. The real constants are also called as floating point constants. The real constant contains a decimal point and a fractional value.

```
Example, 202.15, 300.00
```

These are the valid real constants in 'C'.

A real constant can also be written as,

```
Mantissa e Exponent
```

For example, to declare a value that does not change like the classic circle constant PI, there are two ways to declare this constant

1. By using the **const** keyword in a variable declaration which will reserve a storage memory

   ```
    #include <stdio.h>
   int main() {
   const double PI = 3.14;
   printf("%f", PI);
   //PI++; // This will generate an error as constants cannot be changed
   return 0;}
   ```

2. By using the **#define** pre-processor directive which doesn't use memory for storage and without putting a semicolon character at the end of that statement

   ```
   #include <stdio.h>
   #define PI 3.14
   int main() {
   printf("%f", PI);
   return 0;}
   ```

Data types

'C' provides various data types to make it easy for a programmer to select a suitable data type as per the requirements of an application. Following are the three data types:

1. **Primitive data types**
2. **Derived data types**
3. **User-defined data types**

There are five primary fundamental data types,

1. int for integer data
2. char for character data
3. float for floating point numbers
4. double for double precision floating point numbers
5. void

Array, functions, pointers, structures are derived data types. 'C' language provides more extended versions of the above mentioned primary data types. Each data type differs from one another in size and range. Following table displays the size and range of each data type.

Data type	Size in bytes	Range
Char or signed char	1	-128 to 127
Unsigned char	1	0 to 255
int or signed int	2	-32768 to 32767
Unsigned int	2	0 to 65535
Short int or Unsigned short int	2	0 to 255
Signed short int	2	-128 to 127
Long int or Signed long int	4	-2147483648 to 21474836
Unsigned long int	4	0 to 4294967295
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
Long double	10	3.4E-4932 to 1.1E+4932

Note: In C, there is no Boolean data type.

Integer data type

Integer is nothing but a whole number. The range for an integer data type varies from machine to machine. The standard range for an integer data type is -32768 to 32767.

1 byte=8 bits

2 bytes=16 bits.

An integer typically is of 2 bytes which means it consumes a total of 16 bits in memory. A single integer value takes 2 bytes of memory. An integer data type is further divided into other data types such as short int, int, and long int.

Each data type differs in range even though it belongs to the integer data type family. The size may not change for each data type of integer family.

The short int is mostly used for storing small numbers, int is used for storing averagely sized integer values, and long int is used for storing large integer values.

Whenever we want to use an integer data type, we have place int before the identifier such as,

```
int age;
```

Here, age is a variable of an integer data type which can be used to store integer values.

Floating point data type

Like integers, in 'C' program we can also make use of floating point data types. The 'float' keyword is used to represent the floating point data type. It can hold a floating point value which means a number is having a fraction and a decimal part. A floating point value is a real number that contains a decimal point. Integer data type doesn't store the decimal part hence we can use floats to store decimal part of a value.

Generally, a float can hold up to 6 precision values. If the float is not sufficient, then we can make use of other data types that can hold large floating point values. The data type double and long double are used to store real numbers with precision up to 14 and 80 bits respectively.

While using a floating point number a keyword float/double/long double must be placed before an identifier. The valid examples are,

```
float division;
double BankBalance;
```

Character data type

Character data types are used to store a single character value enclosed in single quotes.

A character data type takes up-to 1 byte of memory space.

Example,

```
Char letter;
```

Void data type

A void data type doesn't contain or return any value. It is mostly used for defining functions in 'C'.

Example,

```
void displayData()
```

Type declaration of a variable

```
int main() {
int x, y;
float salary = 13.48;
char letter = 'K';
x = 25;
y = 34;
int z = x+y;
printf("%d \n", z);
printf("%f \n", salary);
printf("%c \n", letter);
return 0;}
```

Output:

```
59
13.480000
K
```

We can declare multiple variables with the same data type on a single line by separating them with a comma. Also, notice the use of format specifiers in **printf** output function float (%f) and char (%c) and int (%d).

Summary

- A constant is a value that doesn't change throughout the execution of a program.
- A variable is an identifier which is used to store a value.
- There are four commonly used data types such as int, float, char and a void.
- Each data type differs in size and range from one another

Operators in C, Hierarchy of Operators, Expressions

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10

*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then **−**

Show Examples

Operator	**Description**	**Example**
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.

!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e.,

		0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C

+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2

^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

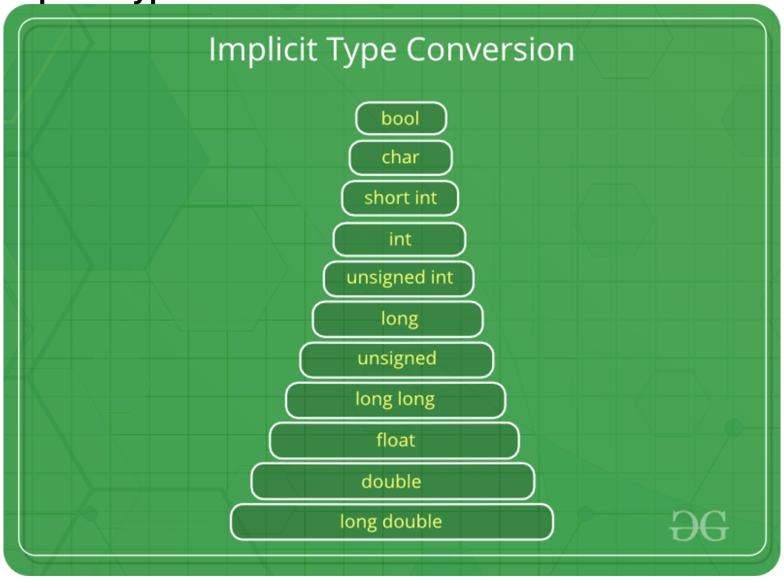
Category	Operator	Associativity

Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Type Conversion in C

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion**



Also known as 'automatic type conversion'.

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

```
bool -> char -> short int -> int ->
unsigned int -> long -> unsigned ->
long long -> float -> double -> long double
```

- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

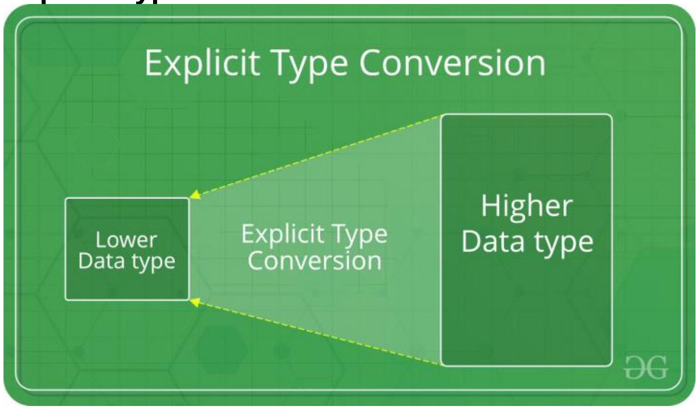
Example of Type Implicit Conversion:

```
// An example of implicit conversion

#include<stdio.h>

int main()

{

    int x = 10;    // integer x

    char y = 'a';  // character c



    // y implicitly converted to int. ASCII

    // value of 'a' is 97

    x = x + y;
    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output:

```
x = 107, z = 108.000000
```

2. **Explicit Type Conversion–**



This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

```
(type) expression
```

Type indicated the data type to which the final result is converted.

```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;


    printf("sum = %d", sum);


    return 0;
}
```

Output:

```
sum = 2
```

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

Library Functions

Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library. Each library function in C performs specific operation. We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.
C Standard **library functions** or simply **C Library functions** are inbuilt **functions in C** programming.
...
Library Functions in Different Header Files.

C Header Files

<**math**.h>	Mathematics functions
<setjmp.h>	Jump functions
<signal.h>	Signal handling functions
<stdarg.h>	Variable arguments handling functions