CHAPTER-6

(OVERVIEW OF C PROGRAMMING LANGUAGE)

Introduction to C

C programming language was developed in 1972 by Dennis Ritchie at Bell Laboratories of AT & T (American Telephone & Telegraph),USA. It was called C because many of its features were derived from an earlier language called 'B'.

BCPL(Basic Combined Programming Language) by Martin Richards in 1967

|

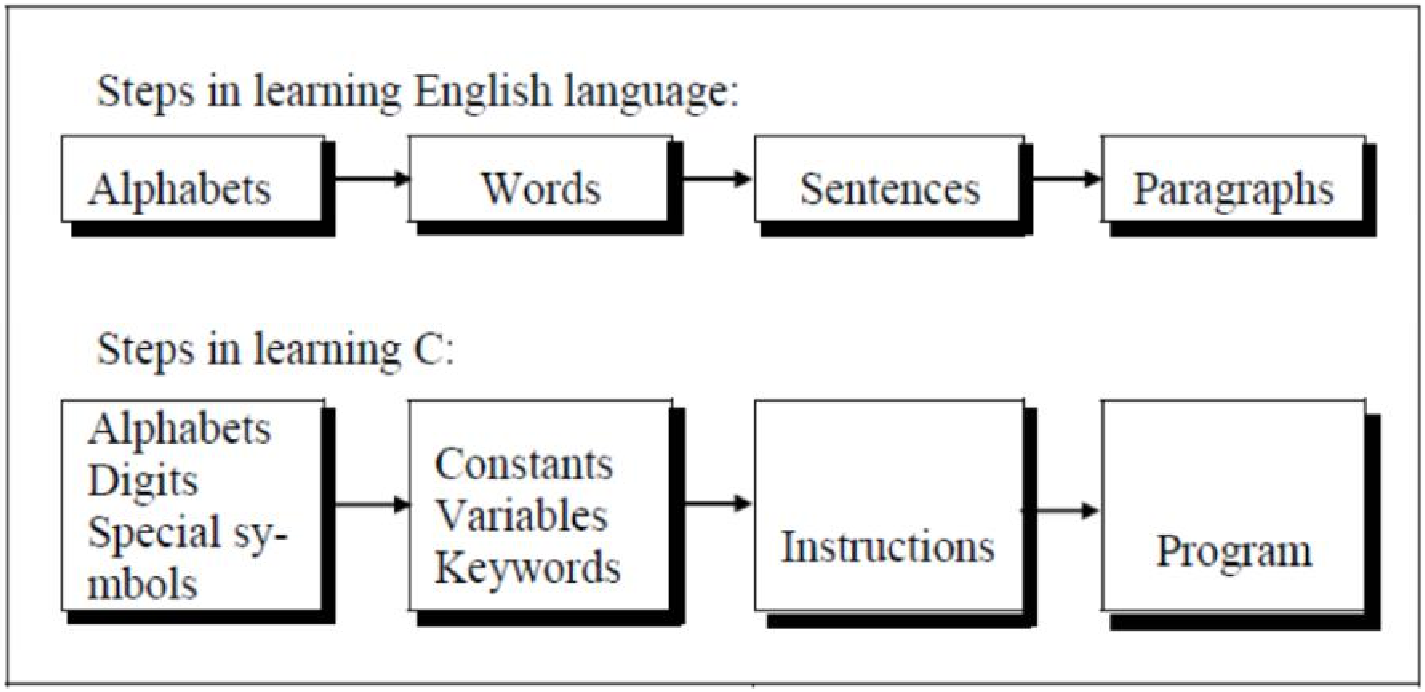
B by Ken Thomson & Dennish Ritchie in 1969

|

C by Dennis Ritchie in 1972

C was initially designed for programming UNIX operating system. Now the software tool as well as C compiler is written in C. Major parts of operating systems like Windows, UNIX, Linux are still written in C. This is because even today when it comes to performance(speed of execution) nothing beats C. If one is to extend the operating system to work with new devices one needs to writes device driver programs. These programs are exclusively written in C. C seems to be popular because, it is simple, reliable and easy to use. C has been superceded by languages like, C++, C#, java.

The general method for learning any language is to learn its alphabet first, then learn to combine these alphabets toform words, which in turn are combined to form sentences and sentences are combined to form paragraph. Learning of C language is very much simmilar. Instead of learning straight-away how to write programs, first of all we have to learn what is alphabets, numbers, special symbol used in C, then using them how contants, variables and keywords are constructed and finally how these are combined to form instruction. A group of instructions are cobined later on to form program.


Steps in learning English language:
Alphabets
Words
Sentences
Paragraphs
Steps in learning C:
Alphabets Digits Special sy-mbols
Constants Variables Keywords
Instructions
Program

Thus a computer program is a collection of the instruction necessary to solve a specific problem. The approach or method that is used to solve the problem is known as algorithm.

So far programming language concerns are of three types.

1) Machine level language
2) Assembly level language
3) High level language

In Machine level language, programs are written in binary language i.e in the form of 0 and 1. These programs are directly understood by the computer. This type of program is not portable, difficult to maintain and prone to error. No translator is required for this program.

In Assembly level language, programs are written using mnemonics or symbol like ADD, SUM, MOV etc. This language is easy to write and understand by the programmer but does not understand by the Computer. So the translator used here is called assembler, which translate assembly level language into machine level language, so that it is understood by the computer. This language is also not portable.

High level language programs are written English like language. It is easy to write and understandable. This language is machine machine independent means it is portable. A translator is required to translate High level language into Machine level language. There are two types of tranlator for High level language: Compiler and Interpreter. The program written in high level language is known as source code and corresponding machine level language is called machine code or object code. Compiler read the program at-a-time and searches for error and lists them. If the program is error free then it is converted into object code. Where as interpreter read one line of the source code and converted into machine code if it is error free. Interpreter checks line by line and hence takes more time than compiler.

Integrated Development Environments(IDE)

The process of editing, compiling, running, and debugging programs is often managed by a single integrated application known as an Integrated Development Environment or IDE. An IDE is a tool that provides user interface with compilers to create, compile and execute programs.

Examples: Microsoft Visual Studio code, Turbo C++, Borland C++, DevC++. These provide integrated developement environment with compiler for both C and C++ programming language. Kylix is a popular IDE for developing applications under Linux. Most IDEs also support program development in several different programming languages in addition to C, such as C# and C++, java .NET etc.

Structure of a C program

A C program is composed of

1) Comment line
2) Preprocessor directive
3) Gobal variable declaration
4) Main function or main()
5) Local variable
6) Statements
7) User defined function

Comment line

It indicates the purpose of the program. It is represented as

/*............................ */

Comment line is used for increasing the understandability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program.

Preprocessor directive

Preprocessor commands or preprocessor directives contain specific instruction which indicate how to prepare the program for compilation. One of the most important and commonly used preprocessor is ‘include’ which tells the compiler that to execute a program, some information is needed from specific header file. Some commonly used header file is stdio.h, conio.h, math.h, alloc.h, string.h, stdlib.h etc. The stdio.h (standard input output header file) contains definition and declaration of system defined function such as printf( ), scanf( ), pow( ) etc.

Generally printf() function used to display and scanf() function used to read value.

Global declaration

This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function.

Main()

Every C program has one main() function and execution of a C program begins from the main() function. It is enclosed within the pair of curly braces. The main() function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
main()
{
........
........
........
}
```

The main( ) function return value when it declared by data type as

```
int main( )
{
------------
-----------
return 0
}
```

The main function does not return any value when it is declared as void (means null/empty) as

```
void main(void ) or void main()
{
------------
-----------
}
```

The program execution start with opening brace and end with closing brace.And in between the two braces declaration part as well as executable part is mentioned. And at the end of each line, the semi-colon is given which indicates statement termination.

Local variables:

The variables declared within a function are known as local variables. Their scope or visibility is limited to the function in which it is declared. They must be declared before their use. Variable declaration describes the data that will be used in the function.

Statements

C statements are a group of code that manipulates data to perform specific task. Each C statements end with semi colon.

User defined function

Function is defined a group or block of C statements that are executed together. C program has one main() function and all other functions are called user- defined function. A C program can have any number of function depending on the tasks that to be performed and each function can have any number of statements arranged according to a specific meaningful sequence. User defined function is also enclosed within the pair of curly braces.

```
*   Structure of a C Program  */
Preprocessor directive
Global declaration
main()
{
    Local declaration;
    Statements;
}

function1()
{
Local declaration;
Statements;
}
----------
----------
functionN()
{
`   Local declaration;
Statements;
}
```

Fig: Structure of a C program

Charater set

A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are:

Alphabets	A, B, ….., Y, Z
	a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { }
	[ ] : ; " ' < > , . ? /

These alphabets, digits, special symbols when properly combined to form constants, variables and keywords.

Identifier

Identifiers are user defined word used to name the variables, arrays, functions, structures etc.

Rules for naming identifiers are:

1) Name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.
2) First characters of a name should be alphabet or underscore
3) Name should not be a keyword.
4) They are case sensitive. That means the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.
5) Identifiers are generally given in some meaningful name such as value, net_salary, age, data etc. An ANSI standard compiler recognize 31 characters.

Keyword

There are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case or small letter. These keywords cann't be used as a variable name as it assigned with fixed meaning. Some examples are **auto, break, case, const, char, continue, default, do, double, elase, enum, extern, float, for, goto, int, if, long, register,return, short, signed, sizeof, struct, stati, switch, typedef, union, unsigned, void, volatile, while** etc.

Data type

Data types is a classification that specifies which type of value a variable can store. It is used for declaring variables or functions of different types before their use. The type of a variable determines how much space it occupies in storage and range of data that can be used.

C program has 3 types of data type:

1) **Basic (built-in) data types:** int, char, float, double and void
2) **Derived data types:** array, pointer and reference
3) **User defined data types:** structure, union, enumeration

A variable declared to be of type “int” can be used to contain integer values only i.e values that do not contain decimal places. A variable declared to be of type “float” can be used for storing floating- point numbers (values containing decimal places). The “double” type is the same as type float, only with roughly twice the precision. The “char” data type can be used to store a single character, such as the letter “a”, the digit character “6”, or the symbolic character “@”.

There are two types of type qualifier in c

Size qualifier: short, long

Sign qualifier: signed, unsigned

When the qualifier unsigned is used the number is always positive, and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type. The size and range of data type in C is given below table.

Basic data type	Data type with type qualifier	Size(byte)	Range
char	char or signed char unsigned char	1 1	-128 to 127 0 to 255
int	int or signed int unsigned int short int or signed short int unsigned short int long int or signed long int unsigned long int	2 2 2 2 4 4	-32768 to 32767 0 to 65535 -32768 to 32767 0 to 65535 -2147483648 to 2147483647 0 to 4294967295
float	float	4	3.4E-38 to 3.4E+38
double	double	8	1.7E-308 to 1.7E+308
long double	long double	10	3.4E-4932 to 1.1E+4932

Constants

Constants are any values that do not change during the execution of a program. Variables may change their value at any time whereas constants never change their value.

Types of constants

1. Integer Constants
2. Floating point or Real Constants
3. Character Constants
4. String Constants
5. Symbolic Constants

Integer Constants

As the name itself suggests, an integer constant is an integer with a fixed value, that is, it cannot have fractional value. It could either be positive or negative. There are three types of integer contants namely:

i) **Decimal Integer Constants**

It has the base/radix 10. (0 to 9)

For example, 55, -20, 1, 220, etc.

ii) **Octal Integer Constant**s

It has the base/radix 8. (0 to 7). In the octal number system, 0 is used as the prefix.

For example, 034, 087, 011, etc.

Iii) **Hexadecimal integer Constants**

It has the base/radix 16. (0 to 9, A to F).

In the hexadecimal number system, 0x is used as the prefix. C language gives you the provision to use either uppercase or lowercase alphabets to represent hexadecimal numbers.

Example: 0XF9BC, 0X327EF etc.

Floating point or Real Constants

Real constants consists of fractional part in their representation.

For Example: 0.0026, 4155.62, 360.2 etc.

Real number can also be represented by exponential notation. The general form of exponential notation is mantissa exponent. The mantissa is expressed either a real number expressed in decimal notation or integer nothation where as the exponent is an integer number with an optional plus or minus sign.

Example: 6.67X10^-11 is represented as 6.67e-11 or 6.67E-11.

Character Constants

A character constants consists of a single character enclosed in single quotes.

Example: ‘5’, ‘a’, ’@’ .

String Constants

A string constants consists of a sequecnce of character enclosed in double quotes.

Example: “abc”, “1234”, “Hello World”.

Symbolic constant

Symbolic constant is a name that substitute for a sequence of characters and, characters may be integer, character or string constant. These constant are generally defined at the beginning of the program as

#define name value -> here name generally written in upper case for example:

```
#define MAX 10
#define CH ‘b’
#define NAME “sony”
```

Variable

A variable is defined as the meaningful name given to the data sorage location on computer memory. A variable holds value that may change any time.

Declaration of variable:

To declare a variable means to create a memory space for the variable depending the data type used and associate the memory with the variable name. All the variables shoud be declared before their use. The general format of any variable declaration

```
datatype v1, v2, v3, ......vn;
```

where v1, v2, v3 are variable names.

```
Example: int a;
         char ch;
         float salary;
```

Variable initialization

When we assign any initial value to a variable during the declaration,then it is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as

Datatype variablename=constant;

Example: int a=20;

Or int a;

a=20;

A global variable is a varibale that is declared above the main body of the source code, outside all function, while local variable is a varibale which is declared within the body of a function or a block.

Managing Input-Output(I/O) Operations

The three essential functions of a computer are reading, processing and writing data. Majority of the programs take data as input, and displays the processed data, often known as information or output. I/O operations are useful for a program to interact with users. stdlib is the standard C library for input-output operations. While dealing with input-output operations in C, two important streams play their role. These are:

1.Standard Input (stdin)

2.Standard Output (stdout)

Standard input or stdin is used for taking input from devices such as the keyboard as a data tream.
Standard output or stdout is used for giving output to a device such as a monitor.

In C programming we can use *scanf()* and *printf()* predefined function to read and print data. There exists several functions that have become more or less standard for input and output operations in C. These are collectively known as the standard I/O library.

Each program that uses a standard input/output function must contain the statement

#include<stdio.h>

at the begining. The file **stdio.h** is an abbreviation for standard input-output header file. The instruction #include<stdio.h> tells the compiler to search for a file named stdio.h and place its contents at this point in the program. The contents of the header file become become part of the source code when it is compiled.

Single Character input

The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard). *getchar()* functioncan be used to read a single character.

Syntax:

```
variable_name = getchar();
```

Example:

```
#include<stdio.h>
void main()
{
char title;
 title = getchar();
}
```

Single Character Output

putcar() is used to write one characters at a time.

Syntax:

```
        putchar(variable_name);
```

Example:

```
#include<stdio.h>
void main()
{
char result = 'P';
 putchar(result);
 putchar('\n');
}
```

Formatted input

It refers to an input data which has been arranged in a specific format. A scanf( ) is used to read formatted data from the keyboard then stores the data in specified program variable.

Syntax:

scanf("control string", arg1, arg2, ..., argn);

The format filed is specified by the control string and the arguments arg1,arg2,...argn specifies the address of location where data is to be stored.

Example: **scanf(" %w d ", &num);**

The above statements wait for the user to enter decimal data from the keyboard and stores it at the address(&) variable 'num'. Here the '%' sign denotes the conversion specification, *w* signifies the integer number that defines the field width of the number to be read and 'd' is a format specifier which represents decimal number. Some other format specifiers are

%c	-	Single character
%s	-	strings or sequence of charaters
%f	-	Floating pount number
%u	-	unsigned number.

Formatted output

A printf( ) is used to display the user required information and also print the values of the variable.

Example-1: printf(" Hello World \n");

The above statement displays the following information:

Hello World

Example-2: printf(" \n Sum of two number is: %d ", sum);

The above statement not only displays the following information but also print the values associated with the variable 'sum'. If the varibale sum=12 then

Sum of two number is: 12

in %d , d is decimal format specifier.

Escape sequence:

Backslash character constants or escape sequence are used in output functions. Although they contain two characters, they represent only one character. Given below is the table of escape sequence and their meaning.

Constant	Meaning	Constant	Meaning
\a	Audible Alert	\f	Formfeed
\r	Carriage Return	\n	New line

\b	Backspace	\t	Horizontal Tab
\v	Vertical Tab	\0	Null

Operator

An operator is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operand to perform operation.
C program has a rich set of operators which can be classified as arithmetic operator, assignment operator, increment operator,decrement operator, logical operator, conditional opeator, comma operator, size of , bitwise and others.

1. Arithmatic Operator

Arithmatic operators are used for arithmatic operation like addition, subtraction, multiplication, division etc. Arithmatic operators require minimum two operands to perform operation. In C there are mainly five arithmatic operators.These are given in the table below.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus operator

* Modulus operator is a special operator in C language which evaluates the reminder of the operand after division.
Example: a+b, a-b, x*y, x/y etc. Here a,b,x,y are known as operands.

2. Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement or assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression.
For example,
int x= y; y=5;
int Sum=x+y+z;

3. Increment and Decrement

The increment(++) and decrement(--) operators are one of the unary operators which are very useful in C language. **Unary operator** is one whcih operate on single operand. Increment operator increases the value of variable by one. Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression .The syntax of the operator is given below.

1. ++ variable_name
2. variable_name ++
3. -- variable_name
4. variable_name --

It is again categories into prefix and postfix operator . In the prefix operator the value of the variable is incremented first , then the new value is used, While in postfix operator first assigns the value of the variable on the left then increment the operand.

Example:

```
let y=5;
x= ++y;
```

Similarly in the postfix increment and decrement operator is used in the operation .

And then increment and decrement is perform.

EXAMPLE

```
let x= 5;
y= ++x;
```

In this case the value of the x and y will be 6.

Suppose if you re-write the above statement as

```
x=5;
y= x++;
```

In this case the value of x will be 5 and y will be 6.

4. Relational Operator

Relational operator is used to compared the value of two operand or expressions depending on their relation. C supports the following relational operators.

Operator	Description	Example
>	Greater than	5>4
>=	Greater than or equal to	Mark >= 50
<	Less than	Age<30
<=	Less than or equal to	Mark<=200
==	Equal equal to	score==mark
!=	Not equal to	5!=4

5. Conditional Operator

It sometimes called as ternary operator, since it requires three expressions as
operand and it uses two symbol such as question mark(?) and the colon(:). The syntax for tarnary operator is as follows

exp1 ? exp2 :exp3

Here exp1 is evaluated first. If exp1 is true then exp2 will be evaluated.If exp1 is false then
exp3 will be evaluated.

Example:

```
void main()
{
int a=10, b=2
int s= (a>b) ? a:b;
printf(“value is:%d”);
}
```

Output: Value is:10

6. Logical Operators

Logical operators are used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression.C program has the following logical operators

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Truth table1: logical AND(&&)

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

Truth table1: logical AND(||)

A	B	A||B
0	0	0
0	1	1
1	0	0
1	1	1

Truth table1: logical NOT(!)

A	A!
0	1
1	0

7. Bitwise Operator

Bitwise operator permit programmer to access and manipulate of data at bit level. Bitwise operator operates on each bit of data.Various types bitwise operator used in C program are

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right

These operator can operate on integer and character value but not on float and double. In bitwise operator the function showbits( ) is used to display the binary representation of any integer or character value.

8. Comma Operator

Comma operator is used to link related expressions together. The comma linked expressions are evaluated from left to right.

Example:

```
int i, j, k, l;
for(i=1,j=2;i<=5;j<=10;i++;j++)
```

9. Sizeof Operator

Size of operator is a Unary operator, which gives the size of a operand in terms of byte that occupied in the memory. An operand may be variable, constant or data type. It determines the length of entities, arrays and structures when their size are not known to the programmer. It is also use to allocate size of memory dynamically during execution of the program.

Example:

```
main( )
{
int sum;
float f;
printf( "%d%d" ,size of(f), size of (sum) );
printf("%d%d", size of(235L), size of(A));
}
```

Expressions

An expression is a combination of variables, constants and operators written according to the syntax of C language. Every expression evaluates to a value. Some examples of C expressions are shown in the table below

Algebric Expression	C Expression
a x b - c	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
ab/c	a*b/c
$3x^2 + 2x + 1$	3*x*x+2*x+1

Evaluation of Expressions

C expressions are evaluated using an assignement statement of the form

variable=expresion

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the exression must be assigned values before evaluation is attempted.

Examples of evaluation statements are

```
x=a*b-c
y=b/c*a
```

Typecasting:

Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type conversion
2. Explicit type conversion

Implicit type conversion

When type conversion takes place automatically, it is known as implicity type conversion. Implicit type conversion happens automatically when a value is copied to its compatible data type. During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.

```
#include<stdio.h>
int main(){
        short a=10; //initializing variable of short data type
        int b; //declaring int variable
        b=a; //implicit type casting
        printf("%d\n",a);
        printf("%d\n",b);
}
```

Output:10

10

Arithmetic Conversion Hierarchy

The compiler first proceeds with promoting a character to an integer. If the operands still have different data types, then they are converted to the highest data type that appears in the following hierarchy chart from top to bottom:

char
Int
Unsigned int
Long
Unsigned long
Long long
Unsigned long long
Float
Double
Long double

Consider the following example to understand the concept:

```
#include <stdio.h>
main() {
   int  num = 13;
   char c = 'k'; /* ASCII value is 107 */
   float sum;
   sum = num + c;
   printf("sum = %f\n", sum );}
```

Output:

```
 sum = 120.000000
```

First of all, the c variable gets converted to integer, but the compiler converts **num** and **c** into "float" and adds them to produce a 'float' result.

Important Points about Implicit Conversions

- Implicit type of type conversion is also called as standard type conversion. We do not require any keyword or special statements in implicit type casting.
- Converting from smaller data type into larger data type is also called as **type promotion**. In the above example, we can also say that the value of 'c' is promoted to type integer.
- The implicit type conversion always happens with the compatible data types.

We cannot perform implicit type casting on the data types which are not compatible with each other such as:

1. Converting float to an int will truncate the fraction part hence losing the meaning of the value.
2. Converting double to float will round up the digits.
3. Converting long int to int will cause dropping of excess high order bits.

In all the above cases, when we convert the data types, the value will lose its meaning. Generally, the loss of meaning of the value is warned by the compiler.

Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion. Suppose we have a variable 'result' that stores the division of two operands which are declared as an int data type.

```
 int result, var1=10, var2=3;
result=var1/var2;
```

In this case, after the division performed on variables var1 and var2 the result stored in the variable "result" will be in an integer format. Whenever this happens, the value stored in the variable "result" loses its meaning because it does not consider the fraction part which is normally obtained in the division of two numbers.

To force the type conversion in such situations, we use explicit type casting.

It requires a type casting operator. The general syntax for type casting operations is as follows:

```
(type-name) expression
```

Here,

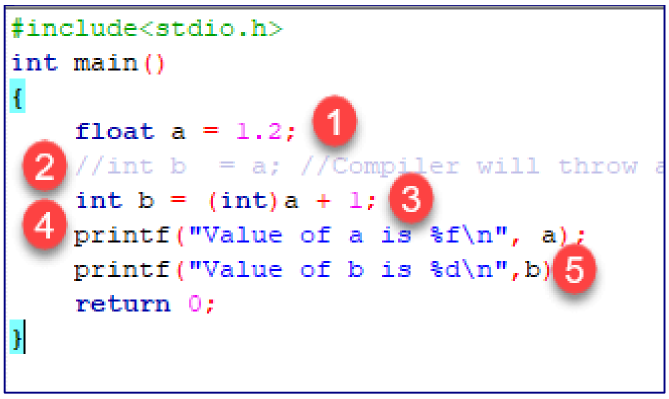
- The type name is the standard 'C' language data type.
- An expression can be a constant, a variable or an actual expression.

Let us write a program to demonstrate implementation of explicit type-casting in 'C'.

```
#include<stdio.h>
int main()
{
        float a = 1.2;
        //int b  = a; //Compiler will throw an error for this
        int b = (int)a + 1;
        printf("Value of a is %f\n", a);
        printf("Value of b is %d\n",b);
        return 0;
}
```

Output:

```
Value of a is 1.200000
Value of b is 2
```



```
#include<stdio.h>
int main()
{
    float a = 1.2;
    //int b  = a; //Compiler will throw a
    int b = (int)a + 1;
    printf("Value of a is %f\n", a);
    printf("Value of b is %d\n",b);
    return 0;
}
```

1. We have initialized a variable 'a' of type float.
2. Next, we have another variable 'b' of integer data type. Since the variable 'a' and 'b' are of different data types, 'C' won't allow the use of such expression and it will raise an error. In some versions of 'C,' the expression will be evaluated but the result will not be desired.
3. To avoid such situations, we have typecast the variable 'a' of type float. By using explicit type casting methods, we have successfully converted float into data type integer.
4. We have printed value of 'a' which is still a float
5. After typecasting, the result will always be an integer 'b.'

Operator Precedence and Associativity

Operator precedence determines which operator is performed first in an expression when more than one operators with different precedence are present. The operators of higher precedence are are evaluated first.

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they evaluated.

Order/Rank	Category	Operator	Meaning of Operation	Associativity
1	Highest precedence	() [] -> .	Function call Array element reference Indirect member selection Direct member selection	Left to Right
2	Unary	! ~ + - ++ -- & * sizeof	Logical negation Bitwise (1's) complement Unary plus Unary minus pre or post increment pre or post decrement Address Pointer reference returns size of operand	Right to Left
3	Multiplicative	* / %	Multiplication Division Modulus	Left to Right
4	Additive	+ -	Binary plus Binary minus	Left to Right
5	Shift	<< >>	Left shift Right shift	Left to Right
6	Relational	< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to Right
7	Equality	==	Equal to	Left to Right

		!=	Not equal to	
8	Bitwise AND	&	Bitwise AND	Left to Right
9	Bitwise XOR	^	Bitwise XOR	Left to Right
10	Bitwise OR	|	Bitwise OR	Left to Right
11	Logical AND	&&	Logical AND	Left to Right
12	Logical OR	||	Logical OR	Left to Right
13	Conditional	? :	Ternary	Right to Left
14	Assignment	= *= %= /= += -= &= ^= |= <<= >>=	Assignment Assignment product Assignment reminder Assignment quotient Assignment sum Assignment difference Assignment bitwise AND Assignment bitwise XOR Assignment bitwise OR Assignment left shift Assignment right shift	Right to Left
15	Comma	,	Comma	Left to Right

Decesion Control and Looping Statements

The **decision control statements** are the decision making statements that decides the order of execution of statements based on the conditions. In decision making statements, the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

C programming language provides the following types of decesion making statements

1 if statement
2. if-else statement
3. nested if statement
4. if-else ladder
5. Switch statement

If statement

The if statement consists a condition which is followed by one or some of the statements, if the condition is true then the statements will be executed or else not.

The syntax for if statement is:

```
if(condition)
{
statements;
}
```

Example: WAP in C to check whether you are eligible for voting or not.

```
#include<stdio.h>
void main()
{
    int age;
    printf("\n Enter your age:");
    scanf("%d", age);
    if(age>=18)
    {
    printf("\n You are eligible for voting");
    }
}
```

if-else statement

The if-else statement has a single condition with two statements block. The syntax of if-else statement is given below:

```
    if(condition)
    {
        statements block1;
    }
    else
    {
        statements block2;
    }
```

Here first of all, condition will be checked. If the condition is true then statements block1 will be executed, but if the condition is false, then statements block2 will be executed.

Example: WAP in C to find the greatest between two numbers.

```
    #include<stdio.h>
    void main()
    {
    int a,b,greatest;
    printf("\n Enter two numers: ")
    scanf("%d %2d", &a,&b);
        if(a>b)
        {
            greatest=a;
        }
        else
        {
            greatest=b;
        }
    printf("\n The greatest number is: %d", greatest);
    }
```

Output:

```
    Enter two numbers: 20 30
```

```
The greatest number is: 30
```

Nested if statement or nested if-else statement

When there are another if else statement in if-block or else-block, then it is called nesting of if-else statement.

Syntax is :-

```
    if(condition-1)
    {
        if(condition-2)
        {
        Statement1;
        }
        else
        {
        statement2;
        }
}
```

Example: WAP in C to find the greatest number among three integer numbers.

```
#include<stdio.h>
void main()
{
    int a,b,c, greatest;
    printf("\n Enter three numbers: ");
    scanf("%d %d %d", &a,&b,&c);
    if(a>b)
    {
        if(a>c)
        {
            greatest=a;
        }
        else
        {
            greatest=c;
        }
    }
  else
    {
        if(b>c)
        {
            greatest=b;
        }
        else
        {
            greatest=c;
        }
    }
printf("\n Greatest number is: %d", greatest);
}
```

Output:

```
        Enter three numbers: 20 30 10
        Greatest number is: 30
```

if-else ladder

In this type of nesting there is an if else statement in every else part except the last part. If condition is false control pass to block where condition is again checked with its if statement.

Syntax is :-

```
        if(condition1)
                Statement1;
        else if(condition2)
                statement2;
        else if(condition3)
                statement3;
                ..............
                ..............
        else if(condition_n)
                statement_n;
        else
                default_statement;
```

This process continue until there is no if statement in the last block. if one of the condition is satisfy the condition other nested "else if" would not executed. But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all condition are checked.

Switch statement

A switch statement or switch case statement is a multi-way decesion statements which is a simplified version of if-else-if statements that evaluates only one variable. It allows us to execute one statements blocks among many alternative. The syntax of swich-case statement is given below

```
switch(variable)
{
        case value1:
                statement block1;
                break;
        case value2:
                statement block2;
                break;
                .........
                .........
        case value_N:
                statement block_N;
                break;
        default:
                default_statement block;
                break;
}
```

The switch case statement compares the value of the variable given in the switch statement with value of each case statement that follows. When the value of the switch and case statement matches, the statement block of that particular case is executed.

The break statement must be used at the end of each case because if it were not used then all the cases from one to N will be executed. The break statement is used to break out the case statement.

Default is also a case that is executed when the value of the variable does not match with value of the case statement.

Example: WAP to demonstrate the use of switch-case statement.

```
#include<stdio.h>
void main()
{
    char grade;
    printf("\n Enter your grade:");
    scanf("%c",grade);
    switch(grade)
    {
    case 'A':
        printf("\n Outstandiang");
        break;
    case 'B':
        printf("\n Exellent");
        break;
    case 'C':
        printf("\n Good");
        break;
    case 'D':
        printf("\n Pass");
        break;
    case 'F':
        printf("\n Fail ");
        break;
    Default:
        printf("\n Invalid grade. Please enter your grade again");
        break;
        }
}
```

Output:

```
    Enter your grade: B
    Exellent
```

Looping statements

Looping or iteration means repeate the execution of group of statements multiple times or until a particular condition is being satisfied. C language supports three types of looping statements

1. **While loop**
2. **do while loop**
3. **for loop**

while loop

while loop provides a mechanism to repeate the execution of one or more statements while a particular condition is true.

The general syntax is:

```
    while(condition)
    {
    statement block;
    }
```

Here condition may be any expression, if it is true the body of the loop will be executed, if it is false control will be come out of the loop.

Example: WAP in C to print " Welcome to C" 5 times.

```
#include<stdio.h>
void main()
{
    int n=1;
    while(n<=5)
    {
    printf(" \n Welcome to C");
    n=n+1;
    }
}
```

```
Output: Welcome to C
        Welcome to C
        Welcome to C
        Welcome to C
        Welcome to C
```

do while loop

In do while loop, the body of the statements is executed first, then condition is checked. If the condition is true, again the body of the statements is executed and this process will continue until the condition becomes false.

There is minor difference between while and do while loop. while loop test the condition before executing any of the statement of block whereas do while loop test condition after having executed the statement at least once. If initial condition is false while loop would not executed it’s statement on other hand do while loop executed it’s statement at least once even If condition fails for first time. It means do while loop always executes at least once.

The syntax for do while loop is

```
    do
    {
    statement block;
    }while(condition);
```

Example: WAP in C to calculate the sum of first N number using do while loop.

```
#include<stdio.h>
void main()
{
    int N,i=0,sum=0;
    printf(“\n Enter the value of N: ”);
    scanf(“%d”, &N);
    do
    {
    sum=sum+i;
    i=i+1
    }while(i<=N);
    printf(“\n The sum of first N number is: %d”, sum);
}
```

for loop

In a program, for loop is generally used when number of iteration are known in advance. The body of the loop can be single statement or multiple statements. It repeates the execution of a bolcks of statements until a particular condition is satisfied.

The general syntax of for loop is

```
    for(initialisation; test condition; increment/decrement)
    {
    statement block;
    }
```

Example: WAP in C to calculate the sum of first N number using for loop.

```
#include<stdio.h>
void main()
{
    int N,i,sum=0;
```

```
        printf("\n Enter the value of N: ");
        scanf("%d", &N);
        for(i=1;i<=N;i++)
        {
        sum=sum+i;
        }
        printf("\n The sum of first N number is: %d", sum);
}
```

Nested loop

Nested loop means a loop can be placed inside other loop. Although this feature will work with any loops but it is most commonly used with for loop.

```
        for(initial valu1; test condition1; increment/decrement)
          {
              for(intial value2; test condition2; increment/decrement)
              {
                inner body of the loop;
              }
              outer body of the loop;
         }
```

For one value of outer loop, inner loop will repeat upto test condition2.So inner loop will be completed first and then outer loop will be completed.

Example: WAP in C to print the following pattern.

```
*    *    *    *    *
*    *    *    *    *
*    *    *    *    *
*    *    *    *    *
```

```
#include<stdio.h>
void main()
{
        int i,j;
        for(i=1;i<=4;i++)
                for(j=1;j<=5;j++)
                printf("\n  *);


}
```

Example: WAP in C to print the following pattern.

```
*
*    *
*    *    *
*    *    *    *
*    *    *    *    *
```

```
#include<stdio.h>
void main()
{
        int i,j;
        for(i=1;i<=5;i++)
                for(j=1;j<=i;j++)
                printf("\n  *);
}
```

Jumping statement

There5 are three different controls used to jump from one part of the program(or statement) to another and make the execution of the programming procedure fast. These are

a) goto statement
b) break statement
c) continue statement

goto statement:

The goto statement moves the controls on a specified address called label or label name. The goto is mainly of two types: one is conditional and other is unconditional. Also jump can be either forward direction or in backward direction.

i) **Forward goto**

In this control moves forward at a specified label either according to condition or without condition

The syntax is

```
        s1;
        s2;
        goto label;
        s3;
        s4;
        label:
        s5;
        s6;
```

(Unconditional forward goto)

```
        s1:
        s2;
        if(condition)
        goto label;
        s3;
        s4;
        label:
        s5;
        s6;
```

(Conditional forward goto statement)

Note that here first statement s1 and s2 will be executed, then specified label and execute the statement s5 and s6. It will skip a part of the program i.e s3 and s4 statements.

ii) **Backward goto**

In backward goto , control moves back to the specified address and so create a loop. In case of conditional backward statement, it creates finite loop and in case of unconditional goto, it creates infinite loop. The general syntax is as follows

```
    s1;
    label:
    s2;
    s3;
    goto label;
    s4;
```

(unconditional backward goto statement)

```
    s1;
    label:
    s2;
    s3;
    if(condition)
    goto label;
    s4;
```

(conditional goto statement)

```
#include<stdio.h>
void main()
{
int a=20,b=30;
printf("\n value of a:", a);
goto mm;
printf("\n value of b:", b);
mm:
printf("\n End of the program");
}
```

Break statement(break)

Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in which it appears and it is generally written with condition. It is written with the keyword as break. When break statement is encountered loop is terminated and control is transferred to the statement, immediately after loop. This break statement is usually associated with if statement.

Example :

```
#include<stdio.h>
void main()
{
```

```
int j=0;
for(;j<6;j++)
if(j==4)
break;
}
```

```
Output:
0 1 2 3
```

Continue statement(continue)

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program. The difference between break and continue is, when the break encountered loop is terminated and it transfer to the next statement and when continue is encounter control come back to the beginning position.
In while and do while loop after continue statement control transfer to the test condition and then loop continue where as in, for loop after continue control transferred to the updating expression and condition is tested.

Example:-

```
void main()
{
    int n;
    for(n=2; n<=9; n++)
    {
    if(n==4)
    continue;
    printf("%d", n);
    }
}
Printf("out of loop");
}
```

```
Output: 2 3 5 6 7 8 9 out of loop
```

CHAPTER-7
(Advanced features of C)

Function

C programs enables the user to devide the large programs into smaller programs known as segments, blocks or modules. Each of which can perform independent task. Each can be coded independently. These are known as functions. It is a good practice to break the large programs into small units based on the task or function.

A function is a self contained block of codes or sub programs with a set of statements that perform some specific task or coherent task when it is called. A C program contains at least one function i.e main() and the exection of program starts from the main function. There are another two types of function used in C programs

1. Built-in function or Library function
2. User-defined function.

Library functions are part of the compiler package. They are predefined or system defined function and cannot be modified. It can only be read and used. Source of these library function are pre-compiled and only object code can get linked with library function by the linker.

User-defined functions are used according to the requirement of the program and created by the programmer himself.

In C user defined function should be declared before its usage in the program

Declaration of function

Function declaration is a statement that identifies a function with its name, lists of parameters and the type of data it returns. The declaration is also called prototype. So function prototype is a declarative statements that tells the compiler about the name, lists of parameter and return type of the function.

The general syntax:

data-type function-name(data-type variable1, data-type variable2,...);

return-type function-name(arguments list/parameters);

Example:

int volume(int a, int b);

***arguments or parameters**

Input that the function takes are known as arguments or parameters. Arguments are placed in the parenthesis. A function can have integers, float, character as an arguments. If the function does ot use any arguments, the word ‘void’ is used within the parenthesis.

***function-name:**

It is defined as any legal identifier followed by parenthesis without any space.

***return-type:**

It is the type of value to be returned by the function. Every function has a return type. If a function does not return a value, its return type will be 'void'.

Function definition

Function definition consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

Syntax:

```
return-type function-name( formal argument list)
{
        statements;                /* Body of the function* /
}
```

Accessing a Function

A function can be accessed or called or invoked from the main program by using function name, followed by parameters enclosed in parenthesis separated by commas.

Example:

```
function(arg1,arg2,arg3);
```

The arguments that are used inside the function call are called **actual arguments.**

Actual argument

The arguments which are mentioned or used inside the function call is knows as actual argument and these are the original values and copy of these are actually sent to the **called function.**

It can be written as constant, expression or any function call like

```
Function (x);
Function (20, 30);
Function (a*b, c*d);
Function(2,3,sum(a, b));
```

Formal Arguments

The arguments which are mentioned in function definition are called formal arguments or dummy arguments. These arguments are used to just hold the copied of the values that are sent by the calling function through the function call. These arguments are like other local variables which are created when the function call starts and destroyed when the function ends. The basic difference between the formal argument and the local variables are

1) The formal argument are declared inside the parenthesis where as the local variable declared at the beginning of the function block.
2). The formal argument are automatically initialized when the copy of actual arguments are passed while other local variable are assigned values through the statements.

Order number and type of actual arguments in the function call should be match with the order number and type of the formal arguments.

Return type

It is used to return value to the calling function. It can be used in two way as

return

Or

Ex:-

```
        return(expression);
        return (a);
        return (a*b);
        return (a*b+c);
```

Here the 1 st return statement used to terminate the function without returning any value.

Ex:- WAP in C to find the summation of two numbers.

```
#include<stdio.h>
int sum(int , int );
void main()
{
int a,b, S;
printf("\n Enter two numners");
scanf("%d %2d",&a,&b);
S=sum(a,b);                         /* function call  */
printf("summation of two numbers is = %d",s);
}
int sum(int x,int y)
{
int z=x+y;
Return z;
}
```

Passing parameters to the function

There are two ways by which we can pass the parameters to the function such as **call by value** and **call by reference.**

1. **call by value**

In the call by value, copy of the actual argument is passed to the formal argument and the operation is done on formal argument.

When the function is called by 'call by value' method, it doesn't affect content of the actual argument. Changes made to formal argument are local to block of called function so when the control back to calling function the changes made is vanish.

Example: WAP in C to swap or interchange values of two number by call by value method.

```
#include <stdio.h>
void swap(int, int);
void main()
{
int x, y;
printf("\n Enter the value of x and y ");
scanf("%d %d",&x,&y);
printf("\n Before Swapping \n x = %d and  y = %d ", x, y);
swap(x, y);
printf("\n After Swapping\n x = %d and y = %d", x, y);
}
void swap(int a, int b)
{
int temp;
temp = a;
a= b;
b = temp;
printf("\n Values of a and b is %d  %d",a,b);
}
```

2. **Call by reference**

Instead of passing the value of variable, address or reference is passed by calling function to called function and the function operate on address of the variable rather than value.

Here formal argument is alter to the actual argument, it means formal arguments calls the actual arguments.

Example: WAP in C to swap or interchange values of two number by call by reference method.

```
#include <stdio.h>
void swap(int *, int *);
void main()
{
int x, y;
printf("\n Enter the value of x and y ");
scanf("%d %d",&x,&y);
printf("\n Before Swapping \n x = %d and  y = %d ", x, y);
swap(&x,&y);
printf("\n After Swapping\n x = %d and y = %d", x, y);
}
void swap(int *a, int *b)
{
int temp;
temp = *a;
*a= *b;
*b = temp;
}
```

Function Recursion

When function calls itself again and again then it is called as recursive function or function recursion. In recursion calling function and called function are same. According to recursion problem is defined in term of itself. Here statement with in body of the function calls the same function.

It can be defined as

```
void recursion()
```

```
    {
        recursion();            /* function calls itself  */
    }
    void main()
    {
    recursion();
    }
```

The C programming language supports recursion, i.e function to call itself. But while using recursion, programmers need to be careful to define exit condition from the function, otherwise it will go into infinite loop. Recursive functions are very useful to solve many mathmatical problems, such as calculating the factorial of a number, generating fibonacci series etc.

Example: WAP in C to find the factorial of a number using function recursion.

```
#include<stdio.h>
int factorial(int );
int main() {
   int n;
   printf("\n Enter a positive integer: ");
   scanf("%d",&n);
   printf("Factorial of %d = %ld", n, factorial(n));
   return 0;
}
 int factorial(int n)
 {
   if (n==1)
      return 1;
   else
      return(n*factorial(n-1));
}
```

Output:

```
Enter a positive integer: 5
Factorial of 5 = 120
```

Example: WAP in C to find the fibonacci series using function recursion.

```
#include<stdio.h>
int Fibonacci(int);
int main()
{
int n, i = 0, c;
scanf("%d",&n);
printf("Fibonacci series\n");
for ( c = 1 ; c <= n ; c++ )
{
printf("%d\n", Fibonacci(i));
i++;
}
return 0;
}
int Fibonacci(int n)
{
if ( n == 0 )
return 0;
else if ( n == 1 )
return 1;
else
return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Output:

Storage Classes

Storage class in C language is a specifier which tells the compiler where and how to store variables, its initial value and scope of the variables in a program. In compiler point of view a variable identify some physical location within a computer where its string of bits value can be stored is known as storage class.

The kind of location in the computer, where value can be stored is either in the memory(RAM) or in the register. There are various storage class which determined, in which of the two location value would be stored.

Syntax of declaring storage classes is:-

storageclass datatype variable name;

There are four types of storage classes and all have keywords:-

1) Automatic (auto)

2) Register (register)

3) Static (static)

4) External (extern)

Examples:-

```
auto float x; or float x;
extern int x;
register char c;
static int y;
```

Compiler assume different storage class based on:-

1) **Storage class:-** tells us about storage place(where variable would be stored).

2) I**ntial value :-**what would be the initial value of the variable.

If initial value not assigned, then what value taken by uninitialized variable.

3) **Scope of the variable:-**what would be the value of the variable of the program.

4) **Life time :-** It is the time between the creation and distribution of a variable or how long would variable exists.

1. Automatic storage class

The keyword used to declare automatic storage class is auto.

Its features:-

Storage-memory location

Default initial value:-unpredictable value or garbage value.

Scope:-local to the block or function in which variable is defined.

Life time:-Till the control remains within function or block in which it is defined.It terminates when function is released.

The variable without any storage class specifier is called automatic variable.

2. Register storage class

The keyword used to declare this storage class is register.

The features are:-

Storage:-CPU register.

Default initial value :-garbage value

Scope :-local to the function or block in which it is defined.

Life time :-till controls remains within function or blocks in which it is defined.

Register variable don't have memory address so we can't apply address operator on it. CPU register generally of 16 bits or 2 bytes. So we can apply storage classes only for integers, characters, pointer type. Revision Variable stored in register storage class always access faster than,which is always stored in the memory. But to store all variable in the CPU register is not possible because of limitation of the register pair.

And when variable is used at many places like loop counter, then it is better to declare it as register class.

Example:-

```
main( )
{
register int i;
for(i=1;i<=12;i++)
printf("%d",i);
}
```

3. Static storage class

The keyword used to declare static storage class is **static**.

Its feature are:-

Storage:-memory location

Default initial value:- zero

Scope :- local to the block or function in which it is defined.

Life time:- value of the variable persist or remain between different function call.

Example:-

```
main( )
{
reduce( );
reduce( );
reduce ( );
}
reduce( )
{
```

```
static int x=10;
printf("%d",x);
x++;
}
```

Output:-10,11,12

4. External storage classes

The keyword used for this class is **extern.**

Features are:-

Storage:- memory area

Default initial value:-zero

Scope :- global

Life time:-as long as program execution remains it retains.

Declaration does not create variables, only it refer that already been created at somewhere else. So, memory is not allocated at a time of declaration and the external variables are declared at outside of all the function.

Example:-

```
int i,j;
void main( )
{
printf( "i=%d",i );
receive( );
receive ( );
reduce( );
reduce( );
}
receive( )
{
i=i+2;
printf("on increase i=%d",i);
}
reduce( )
{
i=i-1;
printf("on reduce i=%d",i);
}
```

Output:-i=0,2,4,3,2.

When there is large program i.e divided into several files, then external variable
should be preferred. External variable extend the scope of variable.

Array

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number or subscript.

An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. And number of subscript always starts with zero. One dimensional array is known as vector and two dimensional arrays are known as matrix.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 2 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantage of C Array

1) **Code Optimization:** Less code to the access the data.
2) **Ease of traversing:** By using the for loop, we can retrieve the elements of an array easily.
3) **Ease of sorting:** To sort the elements of the array, we need a few lines of code only.
4) **Random Access:** We can access any element randomly using the array.

Declaration of an array(single dimentional)

Its syntax is

data-type variable-name[size];

Example:

int arr[50];

int mark[100];

The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space. Like for int data type, it occupies 2 bytes for each element and for float it occupies 4 byte for each element etc.

We can represent individual array or value as : int ar[5];

ar[0], ar[1], ar[2], ar[3], ar[4];

Array subscript always start from zero which is known as lower bound and upper value is known as upper bound and the last subscript value is one less than the size of array. Subscript/index can be any integer, integer constant, integer variable, integer expression.

ar[i], ar[i*7],ar[i*i],ar[i++],ar[3];

Initialisation of single -dimensional Array

Initialisation means assigning intial value to the elements of an array. After declaration element of local array has garbage value. If it is global or static array then it will be automatically initialize with zero.

An explicitly it can be initialize that

Data-type array-name[size] = {value1, value2, value3...}

Example: int arr[5]={20,60,90, 100,120}

Here arr[0]=20, arr[1]=60, arr[2]=90, arr[3]=100, arr[4]=120.

20	60	90	100	120
2000	2002	2003.	2004	2005

Accessing elements of an ARRAY

Example- WAP in C to input values into an array and display them.

```
#include<stdio.h>
int main()
{
int arr[5],i;
for(i=0;i<5;i++)
{
printf("enter a value for arr[%d] \n",i);
scanf("%d",&arr[i]);
}
printf("the array elements are: \n");
for (i=0;i<5;i++)
{
printf("%d\t",arr[i]);
}
return 0;
}
```

OUTPUT:

```
Enter a value for arr[0] = 12
Enter a value for arr[1] =45
Enter a value for arr[2] =59
Enter a value for arr[3] =98
Enter a value for arr[4] =21
The array elements are: 12 45 59 98 21
```

Multi-Dimensional Array

Arrays having more than one dimensions are called multi-dimensional arrays. An array having two dimensions is called two-dimensional array. Multi-dimensional array is also known as matrix.

For declaration, 2-dimensional array contains two subscripts and 3-dimensional array contains three subscript and so on.

Its syntax is

data-type array-name[row][column];

Or we can say 2-d array is a collection of 1-D array placed one below the other.

Total no. of elements in 2-D array is calculated as row*column

Example:-

int a[4][5];

Total no of elements=row*column is 4*5 =20

It means the matrix consist of 4 rows and 5 columns.

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Initialisation of Multi-Dimensional Array

Like one-dimensional arrays, two-dimnensional or multi-dimensional arrays can also be initialised.

data-type array-name[row][column]={initial data values};

Example: iint mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};

These values are assigned to the elements row wise, so the values of
elements after this initialization are

Mat[0][0]=11, Mat[0][1]=12, Mat[0][2]=13,
Mat[1][0]=14, Mat[1][1]=15, Mat[1][2]=16,
Mat[2][0]=17, Mat[2][1]=18,Mat[2][2]=19,
Mat[3][1]=20, Mat[3[2]=21, Mat[3][2]=22

In memory map whether it is 1-D or 2-D, elements are stored in one contiguous manner.

Accessing 2-d array /processing 2-d arrays

For processing 2-d array, we use two nested for loops. The outer for loop corresponds to the row and the inner for loop corresponds to the column.

For example: **WAP in C to input values into an array and display them.**

```
#include<stdio.h>
void main()
{
int a[4][5];
printf("\n Enter the elements of two dimensional array");   /*  for reading value:-   */
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n Entered numbers of two dimensional array is: ");    / * For  value:-  */
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
{
printf("%d",a[i][j]);
}
}
}
```

Example: WAP in C to calculate the summation of 10 random numbers

```
#include<stdio.h>
void main()
{
    int a[10],i,sum=0;
    printf("\n Enter 10 random numbers: ");
+    for(i=1;i<=10;i++)
             scanf("%d", &a[i]);
    for(i=1;i<=10;i++)
    {
    sum=sum+a[i];
    }
    printf("\n The sum of 10 random numbers is : %d", sum);
}
```

Strings

Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character.

We can initialize the string as

char name[]={'j','o','h','n','\o'};

Here each character occupies 1 byte of memory and last character is always NULL character. Where '\o' and 0 (zero) are not same, where ASCII value of '\o' is 0 and ASCII value of 0 is 48. Array elements of character are also stored in contiguous memory location.

From the above we can represent as;

j	o	h	n	\0

The terminating NULL is important because it is only the way that the function that work with string can know, where string end.

String can also be initialized as;

char name[]="John";

Here the NULL character is not necessary and the compiler will assume it automatically.

String constant (string literal)

A string constant is a set of character that enclosed within the double quotes and is also called a literal. Whenever a string constant is written anywhere in a program it is stored somewhere in a memory as an array of characters terminated by a NULL character ('\o').

Example – "m"

"Tajmahal"

"My age is %d and height is %f\n"

The string constant itself becomes a pointer to the first character in array.

Example-char crr[20]="Taj mahal";

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
T	a	j		m	a	h	a	l	\0

String library function

There are several string library functions used to manipulate string and the prototypes for these functions are in header file "string.h". Several string functions are

strlen()

This function return the length of the string. i.e. the number of characters in the string excluding the terminating NULL character. It accepts a single argument which is pointer to the first character of the string.

For example-

```
        strlen("suresh");
        It return the value 6.
```

In array version to calculate legnth:-

```
int str(char str[])
{
        int i=0;
        while(str[i]!='\o')
        {
        i++;
        }
return i;
}
```

Example:- /* String length ....*/

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[50];
print("Enter a string:");
gets(str);
printf("Length of the string is %d\n",strlen(str));
}
```

Output:

```
Enter a string: C in Depth
  Length of the string is 8
```

strcmp()

This function is used to compare two strings. If the two string match, strcmp() return a value 0 otherwise it return a non-zero value. It compare the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.

strcmp(s1,s2)

return a value:

<0 when s1<s2

=0 when s1=s2

>0 when s1>s2

The exact value returned in case of dissimilar strings is not defined. We only know that if s1<s2 then a negative value will be returned and if s1>s2 then a positive value will be returned.

For example:

/* String comparison.........................*/

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

```
#include<string.h>
void main()
{
char str1[10],str2[10];
printf("Enter two strings:");
gets(str1);
gets(str2);
if(strcmp(str1,str2)==0)
{
printf("String are same\n");
}
else
{
printf("String are not same\n");
}
}
```

strcpy()

This function is used to copying one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string.
The old content of the destination string str1 are lost. The function returns a pointer to destination string str1.

Example:-

```
#include<stdio.h>
#include<string.h>
void main()
{
char str1[10],str2[10];
printf("Enter a string:");
scanf("%s",str2);
strcpy(str1,str2);
printf("First string:%s\t\tSecond string:%s\n",str1,str2);
strcpy(str,"Delhi");
strcpy(str2,"Bangalore");
printf("First string :%s\t \tSecond string:%s",str1,str2);
}
```

strcat()

This function is used to append a copy of a string at the end of the other string. If the first string is ""Happy " and second string is "Days" then after using this function the string becomes "HappyDays". The NULL character from str1 is removed and str2 is added at the end of str1. The 2nd string str2 remains unaffected.
A pointer to the first string str1 is returned by the function.

Example:-

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

```
void main()
{
char str1[20],str[20];
printf("Enter two strings:");
gets(str1);
gets(str2);
strcat(str1,str2);
printf("First string:%s\t second string:%s\n",str1,str2);
strcat(str1,"-one");
printf("Now first string is %s\n",str1);
}
```

Output

```
Enter two strings: data
Base
First string: database second string: database
```

strrev()

The purpose of this function is to reverse the string. This function takes string variable as its single argument. Here the first character becomes the last and last character becomes first in a string. It is very useful to find out whether a string is palindrome or not.

Syntax:

```
strrev(st);
```

Here 'st' is a string character type variable.

Pointers

Pointer is a variable which can store the address of another variable rather than the value at that location. It is called pointer because it points to a particular location in memory by storing address of that location.

Benefits of using pointers

Below we have listed a few benefits of using pointers:

1. Pointers are more efficient in handling Arrays and Structures.
2. Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
3. It reduces length of the program and its execution time as well.
4. It allows C language to support Dynamic Memory management.

Concept of Pointers

Whenever a **variable** is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value.

Let us assume that system has allocated memory location `80F` for a variable `a`.

Int a=10;

Here 10 --> is a value.

a--> is name of the storage location.

80F--> is the address of the storage location.

We can access the value `10` either by using the variable name `a` or by using its address `80F`.

The question is how we can access a variable using it's address? Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called **Pointer variables**.

A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.

Declaration of C Pointer variable

General syntax of pointer declaration is,

data-type *pointer-name;

Here * before pointer indicate the compiler that variable declared as a pointer.

Example: .   int *p1;   //pointer to integer type

float *p2;   //pointer to float type

char *p3;   //pointer to character type

When pointer declared, it contains garbage value i.e. it may point any value in the memory.

Initialization of C Pointer variable

Pointer Initialization is the process of assigning address of a variable to a **pointer** variable. Pointer variable can only contain address of a variable of the same data type. In C language **address operator** `&` is used to determine the address of a variable. The `&` (immediately preceding a variable name) returns the address of the variable associated with it.

Example:

```
#include<stdio.h>

void main()
{
    int a = 10;
    int *ptr;        //pointer declaration
    ptr = &a;        //pointer initialization
}
```

Pointer variable always point to variables of same datatype. Let's have an example to showcase this:

```
#include<stdio.h>

void main()
{
    float a;
    int *ptr;
    ptr = &a;        // ERROR, type mismatch
}
```

If you are not sure about which variable's address to assign to a pointer variable while declaration, it is recommended to assign a `NULL` value to your pointer variable. A pointer which is assigned a `NULL` value is called a **NULL pointer**.

Using the pointer or Dereferencing of Pointer

Once a pointer has been assigned the address of a variable, to access the value of the variable, pointer is **dereferenced**, using the **indirection operator** or **dereferencing operator** `*`.

```
#include <stdio.h>

int main()
{
    int a, *p;  // declaring the variable and pointer
    a = 10;
    p = &a;     // initializing the pointer

    printf("%d", *p);    //this will print the value of 'a'

    printf("%d", *&a);   //this will also print the value of 'a'

    printf("%u", &a);    //this will print the address of 'a'

    printf("%u", p);     //this will also print the address of 'a'

    printf("%u", &p);    //this will print the address of 'p'

    return 0;
}
```

Points to remember while using pointers

1. While declaring/initializing the pointer variable, `*` indicates that the variable is a pointer.
2. The address of any variable is given by preceding the variable name with Ampersand `&`.
3. The pointer variable stores the address of a variable. The declaration `int *a` doesn't mean that `a` is going to contain an integer value. It means that `a` is going to contain the address of a variable storing integer value.
4. To access the value of a certain address stored by a pointer variable, `*` is used. Here, the `*` can be read as **'value at'**.

```
#include <stdio.h>

int main()
{
    int i = 10;     // normal integer variable storing value 10
    int *a;     // since '*' is used, hence its a pointer variable

    /*
        '&' returns the address of the variable 'i'
        which is stored in the pointer variable 'a'
    */
    a = &i;

    /*
        below, address of variable 'i', which is stored
        by a pointer variable 'a' is displayed
    */
   printf("Address of variable i is %u\n", a);

    /*
        below, '*a' is read as 'value at a'
        which is 10
    */
   printf("Value at the address, which is stored by pointer variable a is %d\n", *a);

   return 0;
}
```

Pointer Expression

Pointer variable can be used in expression. Arithmatic and comarison operations can be performed on the pointer.

If ptrA and ptrB are properly declared and initialised pointers, then following statements are valid
y=*ptrA+*ptrB;
z=*ptrA+ b;

Pointer Arithmatic:

Pointer arithmetic is different from ordinary arithmetic and it is performed relative to the data type(base type of a pointer).

Example:-

If integer pointer contain address of 2000, on incrementing we get address of 2002 instead of 2001, because, size of the integer is of 2 bytes.

Note:-

When we move a pointer, somewhere else in memory by incrementing or decrement or adding or subtracting integer, it is not necessary that, pointer still pointer to a variable of same data, because, memory allocation to the variable are done by the compiler.

But in case of array it is possible, since there data are stored in a consecutive memory location.

Ex:-

```
void main( )
{
static int a[ ]={20,30,105,82,97,72,66,102};
int *p,*p1;
p=&a[1];
p1=&a[6];
printf("%d",*p1-*p);
printf("%d",p1-p);
}
```

Arithmetic operation never perform on pointer are: addition, multiplication and division of two pointer, multiplication between the pointer by any number, division of pointer by any number and addition of float or double value to the pointer.

Arithmatic operations performed in pointer are:-

a) Addition of a number through pointer.

Example

```
int i=60;
int *p;
p=&i;
p=p+2;
p=p+3;
p=p+9;
```

b) Subtraction of a number from a pointer.

Ex:-

```
int i=20;
int *p1=&a;
p1=p1-10;
p1=p1-2;
```

c) Subtraction of one pointer to another is possible when pointer variable point to an element of same type such as an array.

Ex:-

```
int   a[ ]={2,3,4,5,6,7, 8, 0};
```

```
    int *ptr1,*ptr1;
    ptr1=&a[3];
    ptr2=&a[5];
```

Structure

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

Defining a structure:

`struct` keyword is used to define a structure. `struct` defines a new data type which is a collection of primary and derived datatypes.

Syntax:

```
struct structure-name
{
    data-type variable1;            /* data membe1 or member variables  */
    data-type varibale2;            /* data membe2  */
    data-type variable3;        /* data membe3   */
.........
........
data-type variable-n;           /* data member_n   */
}[structure_variables];
```

Here we start with the `struct` keyword, then it's optional to provide structure name, we suggest you to give it a name, then inside the curly braces, we have to mention all the member variables, which are nothing but normal C language variables of different types like `int`, `float`, `array` etc.

After the closing curly brace, we can specify one or more structure variables, again this is optional.

Note: The closing curly brace in the structure type declaration must be followed by a semicolon(`;`).

Example:

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    // F for female and M for male
    char gender;
};
```

Here `struct Student` declares a structure to hold the details of a student which consists of 4 data fields, namely `name`, `age`, `branch` and `gender`. These fields are called **structure elements or members**. Each member can have different datatype, like in this case, `name` is an array of `char` type and `age` is of `int` type etc. **Student** is the name of the structure and is called as the **structure tag**.

Declaring Structure Variables

It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined . **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways:

1) Declaring Structure variables separately

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};

struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
}S1, S2;
```

Here S1 and S2 are variables of structure Student. However this approach is not much recommended.

Accessing Structure Members

Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot `(.)` operator also called **member access** operator.

Example: WAP in C to Enter students data such as name, age, branch, gender.

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

struct Student
{
    char name[25];
    int age;
    char branch[10];
    char gender;          //F for female and M for male
};

int main()
{
    struct Student s1;   /*  s1 is a variable of Student type and
                             age is a member of Student
                         */
    s1.age = 18;

    strcpy(s1.name, "Viraaj");   /*  using string function to add name    */

    /*
        displaying the stored values
    */
    printf("Name of Student 1: %s\n", s1.name);
    printf("Age of Student 1: %d\n", s1.age);

    return 0;
}
Output:
Name of Student 1: Viraaj
Age of Student 1: 18
```

Initialization of structure variable-

Like a variable of any other datatype, structure variable can also be initialized at compile time.

struct student

```
struct Patient

{
    float height;
    int weight;
    int age;
};

struct Patient p1 = { 180.75 , 73, 23 };    //initialization

OR

struct Patient p1;

p1.height = 180.75;     //initialization of each member separately
p1.weight = 73;
p1.age = 23;
```

Array of Structure

We can also declare an array of **structure** variables. in which each element of the array will represent a **structure** variable. **Example :** `struct employee emp[5];`

The below program defines an array `emp` of size 5. Each element of the array `emp` is of type `Employee`.

```
#include<stdio.h>

struct Employee
{
    char ename[20];
    int salary;
};

void main()
{

struct Employee emp[5];
int i, j;

    for(i = 0; i < 3; i++)
    {
        printf("\nEnter %dst Employee record:\n", i+1);
        printf("\nEmployee name:\t");
        scanf("%s", emp[i].ename);
        printf("\nEnter Salary:\t");
        scanf("%d", &emp[i].sal);
    }

    printf("\nDisplaying Employee record:\n");
    for(i = 0; i < 3; i++)
    {
        printf("\nEmployee name is %s", emp[i].ename);
        printf("\nSlary is %d", emp[i].sal);
    }
}
```

Size of structure-

Size of structure can be found out using **sizeof()** operator with structure variable name or tag-name with keyword.

sizeof(struct student); or

sizeof(s1);

sizeof(s2);

Size of structure is different in different machines. So size of whole structure may not be equal to sum of size of its members.

UNION

A Union is a user defined data type that allows to store different data type in the same memory location. A union may consists of different members, but only one member can contain a value at any time. **Unions** are conceptually similar to **structures**. The syntax to declare/define a union is also similar to that of a structure. The only differences is in terms of storage. In **structure** each member has its own storage location, whereas all members of **union** uses a single shared memory location which is equal to the size of its largest data member.

A **union** is declared using the `union` keyword.

Syntax:

union union-name

{

datatype member1;

datatype member2;

};

Like structure variable, union variable can be declared with definition or separately such as

union union-name

{

datatype member1;

datatype member2;

}v1; */ * v1 variable of union */*

or

union union-name s;

Union members can also be accessed by the dot operator with union variable and if we have pointer to union then member can be accessed by using (arrow) operator as with structure.

Example:-

struct student	union student
{	{
int age;	int age;
char name[20];	char name[20];
float salary;	float salary;
};	};

struct student s;   union student s;

Here datatype/member structure occupy 26 byte of location is memory, where as in the union side it occupy only 20 byte.

Accessing a Union Member in C

Syntax for accessing any `union` member is similar to accessing structure members,

```
union test
{
    int a;
    float b;
    char c;
}t;

t.a;    //to access members of union t
t.b;
t.c;
Example:
#include <stdio.h>

union item
{
    int a;
    float b;
    char ch;
};

int main( )
{
    union item it;
    it.a = 12;
    it.b = 20.2;
    it.ch = 'z';

    printf("%d\n", it.a);
    printf("%f\n", it.b);
    printf("%c\n", it.ch);

    return 0;
}
Output:
-26426
20.1999
z
```

As you can see here, the values of `a` and `b` get corrupted and only variable `c` prints the expected result. This is because in union, the memory is shared among different data types. Hence, the only member whose value is currently stored will have the memory.
In the above example, value of the variable `c` was stored at last, hence the value of other variables is lost.

-------- 0 ----------

Prepared by

K.K. Chapeyar, Lecturer (CSE)
Govt. Polytechnic, Nuapada