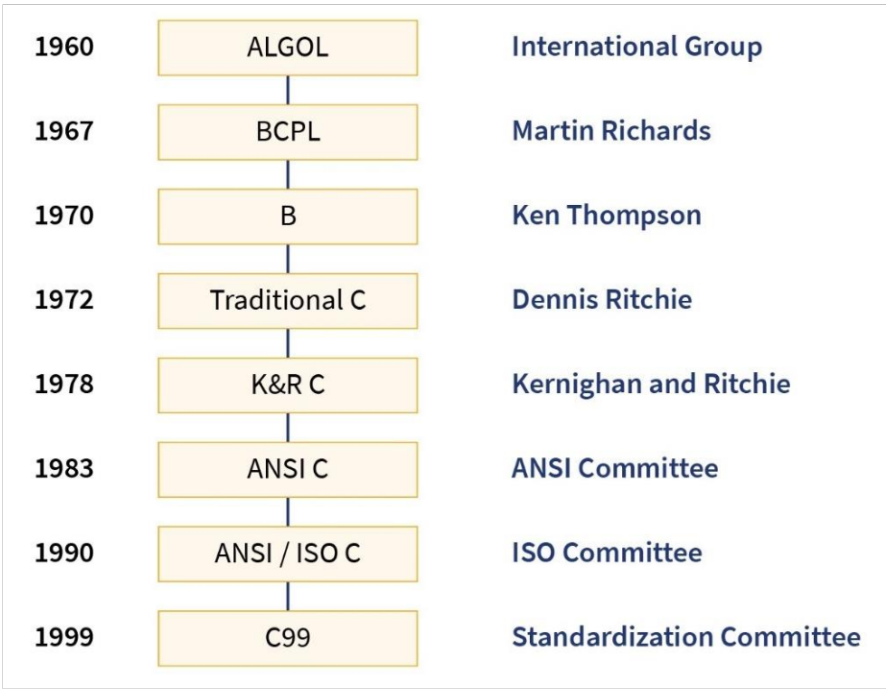
UNIT 2

Introduction to C

History of C language

- ✓ C was evolved from ALGOL, BCPL and B by Dennis Ritchie at the Bell Laboratories in 1972. C uses many concepts from these languages and added the concept of data types and other powerful features.
- ✓ Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system which was also developed at Bell Laboratories, was coded almost entirely in C. UNIX is one of the most popular network operating systems in use today and the heart of the internet data superhighway.
- ✓ For many years, C was used mainly in academic environments, but eventually with the release of many C compilers for commercial use and the increasing the popularity of UNIX, it began to gain widespread support among computer professionals. Today, C is running under a variety of operating system and hardware platforms.
- ✓ During 1970's C had evolved into what is now known as "traditional C".
- ✓ The language become more popular after the publication of the book 'The C programming Language' by Brain Kerningham and Dennis Ritchie in 1978.
- ✓ To assure that the C language remains standard, in 1983, American National Standards Institute (ANSI) appointed a technical committee to define a standard for C. The committee approved a version of C in December 1989 which is now known as ANSI C. It was then approved by the International Standards Organization (ISO) in 1990. This version of C is also referred to as C89.
- ✓ Although C++ and Java were evolved out of C, the standardization committee of C felt that a new features of C++/Java, if added to C, would enhance the usefulness of the language. The result was the 1999 standard for C. This version is usually referred to as C99.
  The history and development of C is illustrated in figure below.



Features of C

The most important features of c programming are listed as follows:

1) **Machine independent**

It is considered as machine-independent language because it allows for writing code that can be compiled and executed on different types of machines with minimal changes.

2) **Middle level language**

C is considered as middle level language because it bridges the gap between low level programming language and high-level programming language by providing features of both high level and low -level language. It can be used to write both system software (eg. Operating system, device drivers) and application software (eg. Games, desktop applications)

3) **Libraries with rich set of functions**

C offers a rich set of libraries with built-in functions for tasks like input/output, memory management, string manipulation, mathematical calculation etc. This makes development faster and easy.

4) **Statically type**

C programming language is a statically typed language. Meaning the type of variable is checked at the time of compilation but not at run time. This means each time a programmer types a program they have to mention the type of variables used.

5) **Fast**

It is well-known that statically typed programming languages are faster than dynamic ones. C is a statically typed programming language, which gives it an edge over other dynamic languages.

Another factor that makes C fast is the availability of only the essential and required features. Newer programming languages have numerous features that increase functionality but reduce efficiency and speed. Since C offers limited but essential features, the headache of processing these features is reduced, resulting in increased speed.

6) **Modularity with structured programming language**
   C is a general-purpose structured language. This characteristic of C language allows you to break code into different parts using functions, which can be stored in libraries for future use and reusability.
7) **Easy to extend**

We can easily extend a C program. If a code is already written, we can add new features with a few alterations.

8) **Pointers**
   Using pointers in C, you can directly interact with memory. As the name suggests, pointers point to a specific location in the memory and interact directly with it.

The C as a middle-level language

- ✓ C is considered as middle level language because it bridges the gap between low level programming language and high level programming language by providing features of both high level and low level language.
- ✓ C can be used to write both system software (eg. Operating system, device drivers) and application software (eg. Games, desktop applications)

Low level features of C language

- ➢ C provides direct access to memory through the use of pointers. This allows programmers to allocate and deallocate memory dynamically using functions like malloc(), calloc(), realloc(), and free().
- ➢ C supports bitwise operators (e.g., &, |, ^, <<, >>) that allow for direct manipulation of individual bits within a data item. This is essential for low-level programming tasks, such as hardware interfacing and data compression.
- ➢ Using low level inline assembly feature of C we can directly access system register.

High level features of C language

- ➢ C syntax is closer to English language, making it more understandable and easier to learn.
- ➢ Program written in C are machine independent and thus portable.
- ➢ C offers a wide range of built-in functions, data types, and operators to facilitate development.
- ➢ Constructs like if-else, for, do-while, and switch enable structured and logical program flow.

The C as a system programming language

C programming language can be used to develop system software such as operating system, device drivers, kernels etc. For example, Linux kernel is written in C. Due to its suitability for writing system software, it is often referred to as a "system programming language. Here is the reason, why c is suited for system programming language.

- ✓ C allows developers to directly access and manipulate memory using pointers. This feature is essential for tasks like memory management, device drivers, and interacting with hardware.
- ✓ C is highly efficient, producing minimal runtime overhead and near assembly-level performance through optimized compilers.
- ✓ C includes a rich standard library that provides functions for I/O, string manipulation, memory allocation, and more. These utilities are crucial for building robust system software.
- ✓ C gives programmers extensive control over the system, from handling interrupts to manipulating hardware registers.

Character set of C

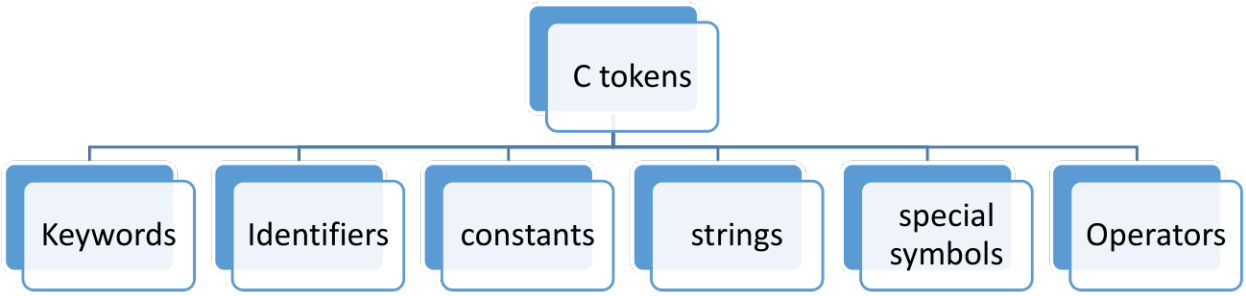
Character set is a package of valid characters recognized by compiler/interpreter. C uses character as building blocks to form basic program elements. (Eg. Constants, variables, expressions etc.

The characters available in C are categorized into following types.

1) **Letters/Alphabets**
   - Uppercase(A-Z)
   - Lowercase(a-z)
2) **Digits** (0-9)
3) **Special symbols** (+,-,*, ! ,%,_,<,>, :)
4) **Whitespace characters**
   - Blank Space
   - Newline
   - Horizontal tab
   - Vertical tab

Tokens

Tokens are the smallest individual units of program. C has six types of tokens .They are as follows:



Keywords

Keywords are predefined words whose meaning has already defined to c compilers. These keywords are used for pre-defined purposes and cannot be used as identifier.The standard keywords are:

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

Identifiers

Identifiers are the names given by user to various program elements such as variables, function and arrays.

Rules of naming Identifiers

- First character must be alphabet (or underscore).
- Must consists of only letters, digits or underscore
- Keyword cannot be used.
- The length of identifiers should not be greater than 31 characters.
- Must not contain whitespace.
- Identifiers are case-sensitive.

Data type

What are the different data types available in C? Explain their types qualifier, conversion character, range of value and storage size in memory occupied by each type.

Definition:

Data type is a classification of type of data that a variable can hold in programming. The data type specifies the range of values that can be used and size of memory needed for that value. Generally data type can be categorized as:

1) **Primary data type**

The size and range of data types on 16 bit machine are:

Data type	Description	Required Memory	Range	Format specifier
char	Used to store character value Eg. 'a','c', 'x'	1 byte	-128 to 127	%c
int	Used to store whole numbers/non fractional numbers. Eg.101, 205, -908	2 bytes	-32768 to +32767	%d
float	Used to store fractional number and has precision of 6 digits. Eg. 5.674, 304.67,67.8903	4 bytes	$-3.4*10^{38}$ to $3.4*10^{38}$	%f
double	Used when precision of float is insufficient and it has precision of 14 digits.	8 bytes	$-1.7*10^{308}$ to $1.7*10^{308}$	%lf
void	Has no values and used as return type for function that do not return a value. Eg. void main()			

2) **Derived data type**
They are derived from existing primary data types.
Eg. Array, Union, Structure

3) **User defined data type**
The data type that are defined by user is known as user defined data types. Examples **enum** (Enumerated data type) and **typedef** (type definition datatype)
General form of typedef

typedef existing_data_type new_name_for_existing_data_type ;

fundamental data type — new identifier

Example
typedef int integer ;
integer symbolizes int data type Now, we can declare int variable "a" as **integer a** rather than **int a .**

Variable

Variable is defined as a meaningful name given to the data storage location in computer memory. It is a medium to store and manipulate data. The value of variable can change during execution of program.

Variable declaration

Naming a variable along with its data type for programming is known as variable declaration.

The declaration deals with two things:

- It tells the compiler what the variable name is
- It specifies what type of data the variable can hold

Syntax:

data_type variable_name ;

eg. int roll;

roll is a variable of type integer. roll can only store integer variable only.

- Explain the necessary rules to define the variable name is C programming.
- Which of the following are invalid variable name and why?

Minimum	First.name	Row total	&name
Doubles	3rd_row	column_total	float

Rules for declaring variables

Variable name must begin with a letter, some system permit underscore as first character.

1. ANSI standard recognizes a length of 31 characters.
2. It should not be keyword.
3. Whitespace is not allowed.
4. Variable name are case sensitive .ie. uppercase and lowercase are different. Thus variable temp is not same as TEMP.
5. No two variables of the same name are allowed to be declared in the same scope.

Variable initialization

Initializing variable means specifying an initial value to assign it.

1) Compile time initialization

In compile time initialization value is assigned to variable using assignment operator '='.

Eg: int a; //variable declaration
a=10; //variable initialization

It is also possible to assign a value at the time of declaration of variable.

Eg. int a=5,b=10;
Float x=7.5;

2) Runtime initialization

In runtime initialization the value is assigned by using scanf() function.

Syntax:

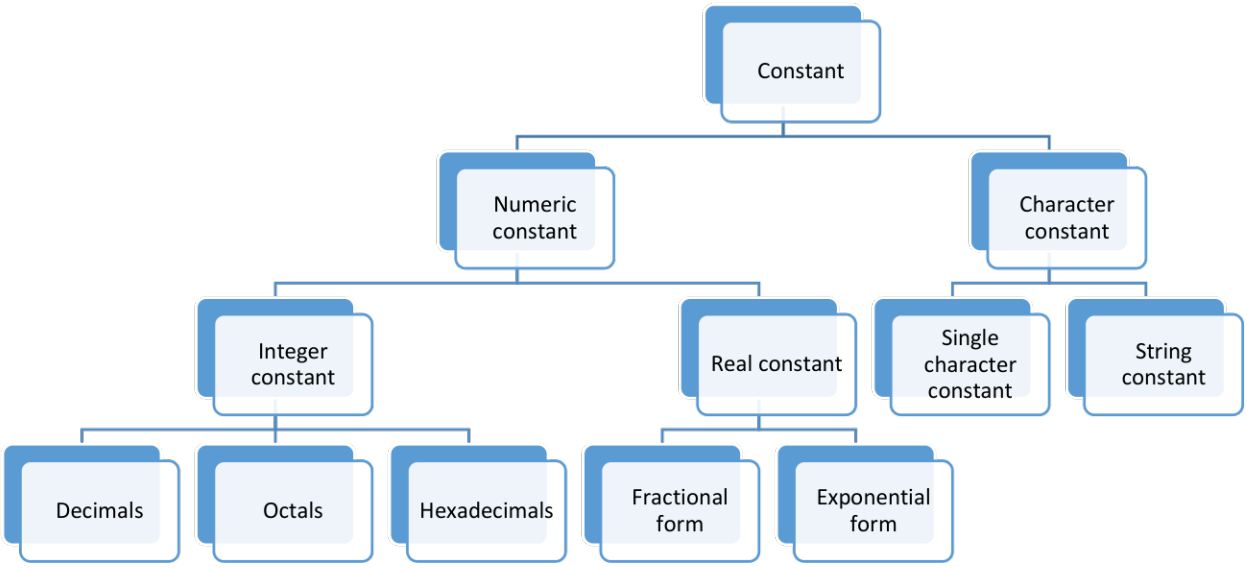
scanf("format specifier",list of address of variable);

eg. scanf("%d%d",&a,&b);

Program to illustrate compile time initialization	Program to illustrate run time initialization
#include<stdio.h> int main() { int a,b; a=5; b=10; printf("value of a=%d\n",a); printf("value of b=%d\n",b); return 0; }	#include<stdio.h> int main() { int a,b; printf("Enter two numbers\n"); scanf("%d%d",&a,&b); printf("value of a=%d\n",a); printf("value of b=%d\n",b); return 0; }

Constants

Constant refers to fixed values that do not change during the execution of program. These fixed values are also called literals. The C constants are as shown in figure. Number associated constant are numeric constant and character associated constant are character constants.



1) Numeric Constant

It consist of:

1.1 Integer constant

It refers to sequence of digits.

There are three types of integer constants.

Decimals: Decimal Integer consist of set of digits, 0 through 9, preceded by an optional +ve or –ve sign.eg.124,-321,678, 654343 etc.

Octal: An octal Integer constant consist of any combination of digits from the set 0 to 7, with a leading 0. eg 037, 075, 0664, 0551

Hexadecimal: Hexadecimal are sequence of digits 0-9 and alphabet A-F with preceding 0x or 0X. eg. 0x5567, 0X53A, 0xFF etc.

1.2. Real or floating point constant

They are numeric constant with decimal points. The real constant are further categorized as

Fractional real constant

They are the set of digits from 0 to 9 with decimal points.eg.394.7867,-0.78676

Exponential real constant

In the exponential form the constants are represented in following form:

mantissa e exponent

where, the mantissa is either a real number expressed in decimal notation or an integer but exponent is always in integer form.

eg. 21565.32=2.156532E4, Where $E4=10^4$

Similarly, -0.000000368 is equivalent to -3.68E-7

2) Character Constant

Single Character Constant

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Eg.

'5' 'X' 'A' '4' etc.

String constant

A string constant is a sequence of characters enclosed in double quotes. The characters may be numbers, special characters and blank space.eg. "Hello" "green" "305" etc.

Type casting

Converting an expression of a given type into another type is known as type casting. Here, it is best practice to convert lower data type to higher data type to avoid data loss.

1**. Implicit Conversion**

Implicit conversions do not require any operator for conversion. They are automatically performed when a value is copied to a compatible data type in the program.

Here, the value of **i** has been promoted from **int** to **float** and we have not had to specify any type-casting operator

.***Program***

```
#include<stdio.h>
int main()
{
 int i=20;
 float result;
 result=i; // implicit conversion
 printf("value=%f", result);
 return 0;
}
```

output

```
Result=20.000000
```

2) Explicit conversion

In this type of conversion one data type is converted forcefully to another data type by the user.

The general rule for cast is

(type-name) expression

Where type-name is one of the standard c data types. The expression may be constant, variable or an expression.

Example	Action
x=(int)7.5	7.5 is converted to integer by truncation
a=(int)21.3/(int)4.5	Evaluated as 21/4 and result would be 5
b=(double)sum/n	Division is done in floating point mode
z=(int)a+b	a is converted into integer and then added to b

Program

```
#include<stdio.h>
int main()
{
 int a=5,b=2;
 float result;
 result=(float)a/b; // Explict conversion
 printf("value=%f",result);
 return 0;
}
```

Output

```
Result=2.50000
```

Escape sequences

Escape sequences are non-printable characters used for formatting the output only. It is a character combination of backslash (\) followed by a letter or by combination of digits. Each sequences are typically used to specify actions such as carriage return, backspace, line feed or move cursors to next line.

The commonly used Escape sequences are as follows

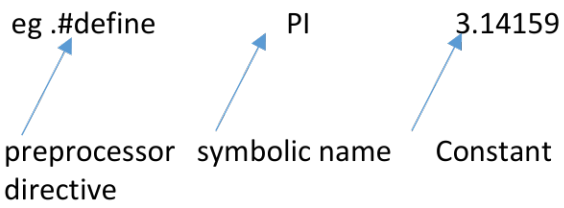
Escape Sequence	Purpose
\a	Alert sound .A beep is generated by a computer on execution.
\b	Backspace
\n	Newline
\r	carriage return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\”	Display double quote character
\’	Display single quote character
\0	Null character. Marks the end of string

Symbolic constant

Symbolic constant is simply a identifier used in place of constants. They are usually defined as the start of the program. When the program is compiled each occurrence of symbolic constant is replaced by corresponding character sequence. Preprocessor directive like #define allow an identifier to have constant value throughout the program.

The syntax for defining symbolic constant is

#define symbolic_constant_name value



Advantages:

- Modifiability
- Understandability

Program to illustrate use of symbolic constant

```
#include<stdio.h>
#define PI 3.1416
int main()
{
float r,area;
printf("Enter the value of r\n");
scanf("%f",&r);
area=PI*r*r;
printf("Area=%f",area);
return 0;
}
```

ASCII

ASCII is the American Standard Code for Information Interchange. It defines an encoding standard for the set of characters and unique code for each character in the ASCII chart.

A character variable holds ASCII value (an integer number between 0 and 127) rather than that character itself in c programming. That value is known as ASCII value.

for example ASCII value of 'A' is 65

what this means is that, if you assign 'A' to a character variable,65 is stored in that variable rather than 'A' itself.

Characters	ASCII values
A-Z	65-90
a-z	97-122
0-9	48-57

Program to print ASCII value of the given character

```
#include<stdio.h>
int main()
{
char ch;
printf("Enter any character\n");
scanf("%c",&ch);
printf("ASCII value of %c is %d",ch,ch);
return 0;
}
```

Input/ Output functions

C programming language provides many built-in functions to read any given input and to display data on screen when there is need to output the result. The header file for input/output functions is <stdio.h>

The input/output function is classified into two types.

1) Formatted I/O functions

2) Unformatted I/O functions

1) Formatted I/O functions

Formatted Input

Formatted Input refers to an input data that can be arranged in a particular format according to user requirements. The built-in function scanf() can be used to input data into the computer from standard Input device.

The general form of scanf is

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

-
- The control string refers to field in which data is to be entered.
- The arguments arg1,arg2 ,…argn specify the address of locations where data is stored.

eg. scanf("%d%d",&num1,&num2);

Formatted output

Formatted output functions are used to display data in a particular specified format.

The printf() is an example of formatted output function.The general form of printf() statement is **printf("control string",arg1,arg2 …… argn);**

The control string may consist of the following data items.

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence characters such as \n,\t and \b

The arguments arg1,arg2,…..argn are the variables whose value are formatted and printed according to specifications of the control string.

Eg. printf("First number=%d\tSecond number=%d",num1,num2);

Example:

```
#include<stdio.h>
int main()
{
      char name[20];
       int age;
       char gender;
       float weight;
       printf("Enter your Name,Age,Gender and Weight\n");
       scanf("%s %d %c %f",name,&age,&gender,&weight);
       printf("Your Name=%s\n",name);
       printf("Your Age=%d\n",age);
       printf("Your Gender=%c\n",gender);
       printf("Your Weight=%0.2f",weight);
       return 0;
}
```

Format specifier

Format specifier is used during input and output. It is the way to tell the compiler what type of data is in variable during taking input using scanf() or printing using printf().In format specifier the percentage(%) is followed by conversion character. This indicates the corresponding data item.

Example:

printf("%d",a);

Here %d is format specifier and it tells the compiler that we want to print an integer value that is present in variable a.

In this way there are several format specifiers in c. Like %c for character, %f for float, etc.

List of different format specifiers

Format specifier	**Purpose**
%c	Character
%d	Signed Integer
%f	Floating point
%u	Unsigned Integer
%o	Octal
%X or %x	Hexadecimal representation of unsigned integer
%e or %E	Scientific Notation of float values
%s	String
%lf	Floating point (double)
%%	Prints % character

2) Unformatted I/O Functions

Unformatted I/O function do not allow the user to read or display data in a desired format.
These type of library functions basically deal with a single character or string of characters.
The functions getchar(), putchar(),gets(),puts(),getch(),getche(),putch() are considered as unformatted functions.

i) getchar() and putchar()

The getchar() reads a character from a standard input device. It buffers the input until the 'ENTER' key is pressed and then assigns this character to character variable.
Syntax is:
character_variable=getchar();
where character_variable refers to some previously declared character variable.
Eg.
char c;
c=getchar();
putchar() function displays a character to standard output device.

Syntax is:
putchar(character_variable);

Program

```
#include<stdio.h>
int main()
{
        char ch;
        printf("Enter the character:");
        ch=getchar();
        printf("Entered character is:");
        putchar(ch);
        return 0;
}
```

Output

```
Enter the character:a
Entered character is:a
```

ii) getch(), getche() and putch()

The function getch() and getche() both reads a single character in a instant. It is typed without pressing the ENTER key. The difference between them is that getch() reads a character typed without echoing it on a screen, while getche() reads the character and echoes (displays) it onto the screen.

Syntax:

```
character_variable=getch();
characher_varibale=getche();
```

The function putch() prints a character onto the screen.

Syntax:
putch(character_variable);

Program

```
#include<stdio.h>
#include<conio.h>
int main()
{
char ch1,ch2;
printf("Enter First character:");
ch1=getch();
printf("\nEnter Second character:");
ch2=getche();
printf("\nFirst character is:");
putch(ch1);
printf("\nSecond  character is:");
putch(ch2);
return 0;
}
```

Output

```
Enter First character:
Enter Second character: b
First character is: a
Second character is: b
```

At first input getch() function read the character input from the user but that the entered character was not displayed. But in second input getche() function read the a character from the user input and entered character was echoed to the user without pressing any key.

iii) gets() and puts()

The gets() function is used to read a string of text containing whitespaces, until newline character is encountered. It can be used as an alternative function for reading strings. Unlike scanf() functions, it does not skip whitespaces(ie.It can be used to read multiwords string)

Syntax: gets(string_variable);

The puts() function is used to display the string on the screen.

Syntax: puts(string_variable);

Program

```
#include<stdio.h>
int main()
{
        char name[20];
        printf("Enter your name:\n");
        gets(name);
        printf("Your name is:");
        puts(name);
        return 0;
}
```

OPERATOR

Operator

- ✓ Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- ✓ Operators are used in program to manipulate data and variables.

Operand

The data items on which operators are act upon are called operands.

eg. a + b

- Here, symbol + is known as operator
- a and b are operands

Expression

The combination of variables, constants and operators written according to syntax of programming language is known as Expression.

eg. a+b*c-5

Types of operators

Types of Operators are classified as:

1. On the basis of no of operands
2. On the basis of function/utility of an operator

1. On the basis of number of operands

i) Unary operator

The operator which operates on single operand known as unary operator.

(++) increment Operator
(--) Decrement operator
+ Unary plus
- Unary minus
eg . ++x, -5, +8 , y- - etc.

ii) Binary operator

The operator which operates on two operands are known as binary operator.

eg. +(addition), – (subtraction) , /(division) , * (multiplication) , <(less than) >(greater than) etc.

eg 3+3, a>b , m*n , x-y

iii) Ternary operators

The Operators that operates on three operands are known as ternary operator.

Ternary operator pair “?:” is available in c .

Syntax:

expression1? expression2:expression3

The expression1 is evaluated first,

- if it is true then expression2 is evaluated and its value becomes value of the expression
- If it is false expression3 is evaluated and its value becomes value of the expression.

Eg .Let us consider the program statement

```
int a=15,b=10,max;
max = (a>b)?a: b;
```

Here, The value of a will assigned to max.

2. On the basics of functions (utility) of an operator

1) Arithmetic Operators

They are used to perform Arithmetic/Mathematical operations on operands.

Operator	Meaning	Example If a=20 and b=10
+	Addition	a+b=30
-	Subtraction	a-b=10
*	Multiplication	a*b=200
/	Division	a/b=2
%	Modulo Division	a%b=0

2) Relational Operators

These are used to compare two quantities and depending on their relation take certain decision. The value of relational operator is either 1(if the condition is true) and 0 (if the condition is false).

Operator	Meaning	Example If a=20 and b=10
==	Equal to	(a==b) is not True
!=	Not Equal to	(a!=b) is True
>	Greater than	(a>b) is True
<	Less than	(a<b) is not True
>=	Greater than or Equal to	(a>=b) is True
<=	Less than or Equal to	(a<=b) is not True

3) Logical Operators

Logical Operators are used to compare or evaluate logical and relational expressions and returns either 0 or 1 depending upon whether expressions results true or false.

Operator	Meaning	Example If a=20 and b=10
&&	Logical AND	Expression ((a==b)&&(a>15)) equals to 0
||	Logical OR	Expression ((a==b)||(a>15)) equals to 1
!	Logical NOT	Expression !(a==b) equals to 1

4) Assignment Operators

Assignment Operators are used to assign the result of an expression to a variable. The mostly used assignment operator is "=".

Operator	Name	Example
=	Assignment	c=a+b will assign value of a+b into c
+=	Add AND assignment	c+=a is equivalent to c=c+a
-=	Subtract AND assignment	c-=a is equivalent to c=c-a
*=	Multiply AND assignment	c*=a is equivalent to c=c*a
/=	Divide AND assignment	c/=a is equivalent to c=c/a
%=	Modulus AND assignment	c%=a is equivalent to c=c%a

5) Increment and Decrement Operator

The increment operators (++) causes its operand to be increased by 1 whereas decrement operator (--) causes its operand to be decreased by 1.

They are unary operators (ie. They operate on single operand).It can be written as

- x++ or ++x (post and pre increment)
- x-- or - - x (post and pre decrement)

Note: The increment and decrement operators can be used as postfix and prefix notations.

Prefix notation: The variable is incremented (or decremented) first and then expression is evaluated using the new value of the variable.

Eg. int m=5;
y=++m;
In this case the value of y and m would be 6.

Postfix notation: The expression is evaluated first using the original value of the variable and then variable is incremented (or decremented by one).

Eg. int m=5;

y=m++;

The value of y would be 5 and m would be 6.

6) Conditional Operator

The ternary Operator pair "?:" is available in c to construct condtional expressions of the form.

expression1? expression2: expression3

The expression1 is evaluated first,

- if it is true then the expression 2 is evaluated and its value becomes the value of the expression.
- If expression1 is false expression3 is evaluated and its value becomes the value of the expression.

Example:

Consider the following statements.

```
int a, b;
a=10;
b=15;
max= (a>b)? a: b;
```

In this example value of b will be assigned to max.

Program to find the maximum number between two numbers using conditional operator.

```
#include<stdio.h>
int main( )
{
        int num1,num2,max;
        printf("Enter any two numbers\n");
        scanf("%d%d",&num1,&num2);
        max=(num1>num2)?num1:num2;
        printf("The largest number = %d",max);
        return 0;
}
```

7) Bitwise operators

The operators which are used for the manipulation of data at bit level. These operators are used for testing the bits, shifting them right or left.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

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	0	0	1	1
1	1	1	1	0

Assume a = 60 and b = 13 in binary format, they will be as follows −

```
        a = 0011 1100
        b = 0000 1101
         ----------------
a&b =    0000 1100      =>12
a|b =    0011 1101      =>61
a^b =    0011 0001      => 49
```

Bitwise shift operators

i.Left shift <<

This causes the operand to be shifted left by some bit positions.

General form:

Operand<<n

Eg.

n=60, n1=n<<3

N	0000 0000	0011 1100
First shift	0000 0000	0111 1000
Second shift	0000 0000	1111 0000
Third shift	0000 0001	1110 0000

After left shifting value of n1 becomes 480

ii. Right shift >>

This causes operand to be shifted to the right by some bit positions.

Operand >>n

Eg. n=60, n2=n>>3

N	0000 0000 0011 1100
First shift	0000 0000 0001 1110
Second shift	0000 0000 0000 1111
Third shift	0000 0000 0000 0111

After Right shifting value of n2 becomes 7

8) Special operators

C supports some special operators. Some of them are

i. Sizeof operator

The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may ne variable, constant or data type qualifier.

Examples:

```
m=sizeof(sum);
n=sizeof(long int);
k=sizeof(235L);
```

Program

```
#include<stdio.h>
int main()
{
        float num;
        printf("Number of bytes allocated =%d",sizeof(num));
        return 0;
}
```

ii. Comma operator

The comma operator can be used to link related expressions together .A comma-linked list are evaluated left to right and the value of right most expression is the value of combined expression.

For example the statement

```
value = (x=10, y=5, x+y);
```

First assigns the value 10 to x then assigns 5 to y and finally assigns 15 (ie 10+5) to value.

Operator precedence and associativity

Write a short note on:

Operator precedence and associativity.

- ✓ Operator precedence is a predefined rule of priority of operators. If more than one operators are involved in an expression the operator of higher precedence is evaluated first.
- ✓ Associativity indicates the order in which multiple operators of same precedence executes.

Precedence	Operator	Associativity
1	(), {}	Left to right
2	++, --, !	Right to left
3	*, /, %	Left to right
4	+, -	Left to right
5	<, <=, >, >=	Left to right
6	== ,!=	Left to right
7	&&	Left to right
8	||	Left to right
9	?:	Right to left
10	=, *=, /=, %=, -=	Right to left

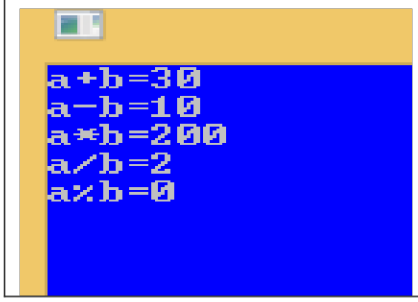
Some operators Examples:

Program to illustrate Arithmetic Operators

```c
#include<stdio.h>
#include<conio.h>
int main()
{

int a=20, b=10, c;
c=a+b;
printf("a+b=%d\n", c);
c=a-b;
printf("a-b=%d\n", c);
c=a*b;
printf("a*b=%d\n", c);
c=a/b;
printf("a/b=%d\n", c);
c=a%b;
printf("a%%b=%d", c);
getch();
return 0;
}
```

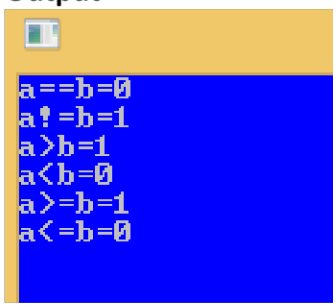
Output



Program to illustrate Relational Operators

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=20, b=10;
    printf("a==b=%d\n", a==b);
    printf("a!=b=%d\n", a!=b);
    printf("a>b=%d\n", a>b);
    printf("a<b=%d\n", a<b);
    printf("a>=b=%d\n", a>=b);
    printf("a<=b=%d\n", a<=b);
    getch();
    return 0;
}
```

Output

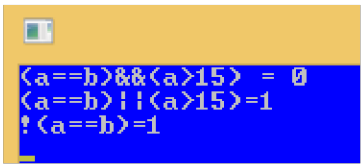


Program to illustrate logical operators

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=20,b=15,result;
    result=(a==b)&&(a>15);
    printf("(a==b)&&(a>15) = %d\n",result);
    result=(a==b)||(a>15);
    printf("(a==b)||(a>15)=%d\n",result);
    result=!(a==b);
    printf("!(a==b)=%d\n",result);
    getch();
    return 0;

}
```

Output

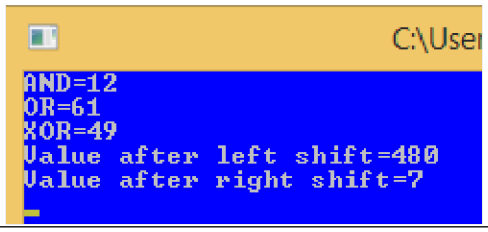


Program to illustrate assignment operators

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int a=5, c;
    c=a;
    printf("c=%d\n", c);
    c+=a;
    printf("c=%d\n", c);
    c-=a;
    printf("c=%d\n", c);
    c*=a;
    printf("c=%d\n", c);
    c/=a;
    printf("c=%d\n", c);
    c%=a;
    printf("c=%d\n", c);
    getch();
    return 0;

}
```

output

Program to illustrate Increment/Decrement Operators	***Program to illustrate Bitwise operators***
`` #include<stdio.h> #include<conio.h> int main() { int a=5,b=10,c=15,d=20,result; result=++a; printf("%d\n",result); result=b++; printf("%d\n",result); result=--c; printf("%d\n",result); result=d--; printf("%d\n",result); getch(); return 0; } ```	``` #include<stdio.h> int main() { int n1=60,n2=13,x,y,z,left,right; x=n1&n2; y=n1|n2; z=n1^n2; left=n1<<3; right=n1>>3; printf("AND=%d\n",x); printf("OR=%d\n",y); printf("XOR=%d\n",z); printf("Value after left shift=%d\n",left); printf("Value after right shift=%d\n",right); return 0; } ```
Output	***output***
	

Data type modifier/Type Qualifiers

Data type modifiers or type Qualifiers are used to modify the properties of primitive or basic data types according to application requirements; so that we can be able to precisely utilize the computer memory.

Modifiers are prefixed with basic data types to modify the size of memory allocated for a variable. As the size varies the range of values stored by the variable also changes.

With the help of data type modifiers, we can:

- ✓ Modify the size(i.e. the amount of memory to be allocated)
- ✓ Modify the sign (i.e decide only +ve or both +ve or –ve values can be stored)
- ✓ Modify the range(i.e .increase or decrease the range of values stored by data type)

  - ✓ **short** and **long** are size modifiers
  - ✓ **signed** and **unsigned** are sign modifiers.

short: short qualifier decreases the size of data type as well the range of values stored by the variable decreases.

long: Long qualifier increases the size of the basic data type as well the range values stored by variable increases

unsigned: Allows to store only +ve values.

signed: Allows to store +ve as well as –ve values. By default primitive data is prefixed with signed

Primitive data type			Data type after using Modifiers		
Type	**Range**	**Size**	**Type**	**Range**	**size**
int	-32768 to +32767	2 bytes	signed int	-32768 to +32767	2 bytes
			unsigned int	0 to 65535	2 bytes
			long int	-2,147,483,648 to 2,147,483,647	4 bytes
			short int	-128 to +127	1 byte
			signed long int	-2,147,483,648 to 2,147,483,647	4 bytes
			unsigned long int	0 to 4,294,967,295	4 bytes
double	1.7E-308 to 1.7E+308	8 bytes	long double	3.4E -4932 to 1.1E +4932	10 bytes
char	-128 to +127	1 byte	signed char	-128 to +127	1 byte
			unsigned char	0 to 255	1 byte

How could you extend the range of the values represented by the basic data types?

OR

How can you modify the size of data type?

Data type modifiers or type Qualifiers are used to modify the properties of primitive or basic data types.

With the help of data type modifiers, we can

- ✓ Modify the size (i.e., the amount of memory to be allocated)
- ✓ Modify the range (increase or decrease the range of values stored by data type)
- ✓ Modify the sign (i.e decide only +ve or both +ve or –ve values can be stored)

So, with the help of data type modifiers we can extend the range of values represented by the basic data types or modify the size of data type.

There are four data type modifiers in C. They are

1. long
2. short
3. unsigned
4. signed

- ✓ Short and long are size modifiers
- ✓ Signed and unsigned are sign modifiers.

For example, storage space for int data type is 2 bytes with range -32768 to +32767 in 16-bit compiler. We can increase the range -2,147,483,648 to +2,147,483,647 by using long int which is of 4 bytes.

Similarly, we can decrease the range -128 to +127 by using short int which is 1 byte.

IMPORTANT:

How can you declare the following variables using suitable data types? Mobile number, Distance jumped by frog, salary and Age. Also explain their type qualifier, memory occupancy size, conversion character and range.

Mobile number

- ✓ As mobile number holds long digits of positive numbers so that **long int** data type will be suitable.
- ✓ **Type qualifier:** mobile number holds only positive and large range of values. So that we can use the combination of unsigned and long i.e **unsigned long** as a type qualifier.
- ✓ **Memory occupancy size:** 4 bytes for 16-bit compiler
- ✓ **Conversion character (format specifier):** %ld
- ✓ **Range** of long int: -2,147,483,648 to +2,147,483,647

Note: similar for registration number, symbol number, account number

Distance jumped by frog

- ✓ The distance jumped by frog is in decimal value so that we can we **float** as a datatype.
- ✓ **Type qualifier:** No any type qualifier can be used with float
- ✓ **Memory occupancy size:** 4 bytes for 16-bit compiler
- ✓ **Conversion character (format specifier):** %f
- ✓ **Range of float:** $-3.4*10^{38}$ to $3.4*10^{+38}$

For more precision of decimal numbers, we can also use double as an datatype. We can use long as an type qualifier to increase the **size** as well **range** of values stored by double.

Note: similar for weight, salary, body temperature

Age

- ✓ As age is non-decimal value, we use **int** as an datatype to store values.
- ✓ **Type qualifier:** As age stores only positive values we can use **unsigned** as a type qualifier.
- ✓ **Memory occupancy size:** 2 bytes for 16-bit compiler
- ✓ **Conversion character (format specifier):** %d
- ✓ **Range of int:** - 32768 to +32767

(Note: Here, we are mentioning memory occupancy size, conversion character,range of data type)

Mention the appropriate data type for storing the following data. Also justify your answer in brief.

A prime number between 5 and 555

As there exist many prime numbers between 5 and 555 the appropriate data type will be array of integers. Array being a collection of similar data elements and derived data type the prime number between 5 and 555 can be stored under a common variable name.

How can you declare following variables using suitable data types? Also explain each memory size and range.

Address

The suitable data type for holding the address is string. As we know strings are sequence of characters stored in consecutive memory location. Each character in string occupies one byte of memory. Strings always terminated with null character '\0'.

So that the memory size required to holding address depends upon the number of characters in address.

Eg. The address named "Kathmandu" requires 10 bytes, including null character.

So, declaration,

char address[10];

is sufficient of hold address named Kathmandu.

As string being sequence of character, the range for each character in string is same as range of character i.e. -128 to +127.

BASIC C PROGRAMS

1) Write a program to find sum of two numbers.

```
#include<stdio.h>
int main()
{
int num1,num2,sum;
printf("Enter any two numbers\n");
scanf("%d%d",&num1,&num2);
sum=num1+num2;
printf("sum of two numbers=%d",sum);
return 0;
}
```

2) Write a program to calculate area and circumference of circle having the radius r should be taken from user.

```
#include<stdio.h>
int main()
{
float r,area,circum;
printf("Enter the radius of circle\n");
scanf("%f",&r);
area=3.1416*r*r;
circum=2*3.1416*r;
printf("Area of circle=%f\n",area);
printf("Circumference of circle is %f\n",circum);
return 0;
}
```

3) Write a program to swap two variables.

```
#include<stdio.h>
int main()
{
int a,b,temp;
a=5;
b=10;
printf("variables before swapping a=%d
and b=%d\n",a,b);
temp=a;
a=b;
b=temp;
printf("variables after swapping a=%d and
b=%d",a,b);
return 0;
}
```

4) **Write a program to check the given number is exactly divisible by but not 7.**

```
#include<stdio.h>
int main()
{
int num;
printf("Enter any number\n");
scanf("%d",&num);
if(num%5==0&&num%7!=0)
{
printf("The given number is exactly divisible by 5
but not 7");
}
else
{
printf("The given number does not satify the
above condtion");
}
return 0;
}
```

5) WAP that will convert temperature in celsius into Fahrenheit.

```
#include<stdio.h>
int main()
{
float cel,fah;
printf("Enter the temperature in Celsius\n");
scanf("%f",&cel);
fah=1.8*cel+32;
printf("The equivalent temperature in fahrenheit
is %f",fah);
return 0;
}
```

6) WAP to check the given 3 digit number is armstrong or not.

```
#include<stdio.h>
#include<math.h>
int main()
{
int num,a,b,c,arm;
printf("Enter the number\n");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
arm=pow(a,3)+pow(b,3)+pow(c,3);
if(num==arm)
{
 printf("Entered number is armstrong number");
}
else
{
printf("Entered number is not armstrong number");
}
return 0;
}
```

7) WAP to find the reverse of a given (3 digit) number.

```
#include<stdio.h>
int main()
{
int num,a,b,c,rev;
printf("Enter any three digit number\n");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
rev=c*100+b*10+a;
printf("Reverse of given number is %d",rev);
return 0;
}
```

8) WAP to find the sum of digit of given (3 digit) number.

```
#include<stdio.h>
int main()
{
int num,a,b,c,sum;
printf("Enter any three digit number\n");
scanf("%d",&num);
c=num%10;
b=(num/10)%10;
a=num/100;
sum=a+b+c;
printf("Sum of digit of given number is %d",sum);
return 0;
}
```