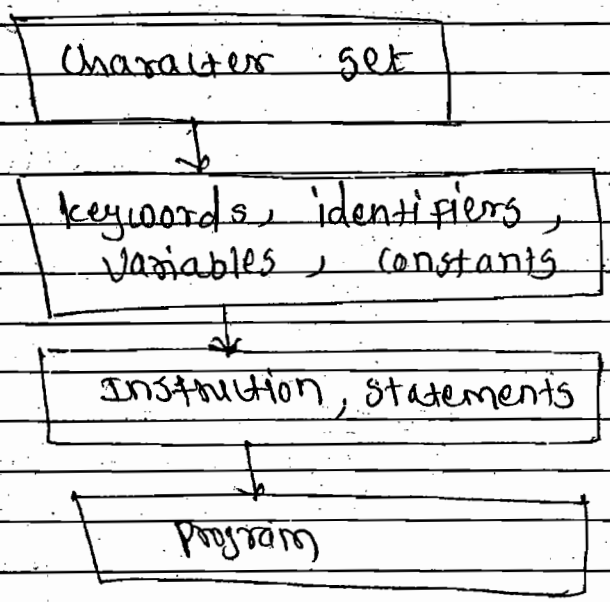
'C' is a general purpose structured programming language designed and written by Dennis Ritchie at AT&T's Bell laboratories of USA in 1972.

2. Basics of C programming

* Data concepts :- While learning any programming language knowing the ~~alphabet~~ Character set is first step. After Character set keywords, identifiers, variables, constants are created using this Character set.
- An instruction is made up of valid set of keywords, identifiers, variable, constants.
- A program is nothing but the set of instructions.



* Character set :- A character indicates any alphabet, digit, or special symbol used to represent the data.

Alphabets	—	A to Z, a ---- z
digits	—	0 to 9
special symbols	—	", !, &, ', ¦, $, #, @, :, <, >, =, +, -, *, /, \, ?, ., ;, :, (,), {, }

* Token :- Whenever we write a paragraph the individual words and punctuation marks are called as tokens.
OR the smallest individual unit in a 'C' program is called as C tokens.

e.g.

```
printf("\n welcome");
```

Here each and every valid unit of a 'C' program is considered as token.

Types of 'C' Tokens :-
1) keywords 2) Identifiers
3) constants 4) Operators
5) strings 6) special symbol

1) keywords :- keywords are the reserved words which C uses for its internal purpose. The meaning of these words are already known to the 'C' compiler.
These keywords we can not used to declare the variables.
There are 32 keywords in 'C' programming

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

- keywords are the special words whose meaning has pre-defined to the 'C' compiler and it can not be changed
- All keywords must be written in lowercase.

~~Calculate total sales and~~

⇒ Identifiers :- Identifier is a collection of alphanumeric characters. Identifiers are used to give name to the programming elements like variable, arrays, functions, structures, unions and labels.
Identifier is formed by combination of uppercase letters, lowercase letters, digits and the underscore symbol.
- Identifier is a user defined word.

e.g. :- sum, area_of_circle, not etc.

* Difference between keywords and identifiers

keywords	Identifier
1) keywords are the predefined word means their meaning is already explained to the compiler	1) Identifiers are user defined means meaning is not explained to the compiler.
2) keywords are combination of alphabetic characters	2) Identifiers are combination of alphanumeric characters and underscore symbol.
3) keywords are written in lowercase only	3) Identifiers can be written in both lower case and uppercase
4) underscore character is not used in keywords	4) underscore character is considered as a letter in identifier.

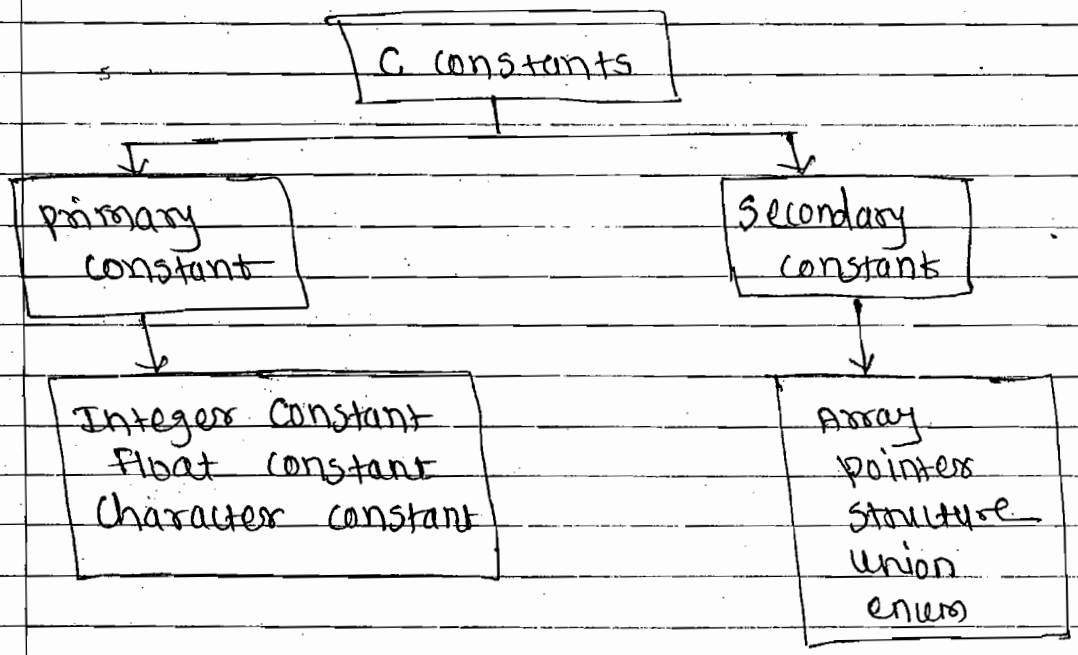
3) Constants :- A constant is a entity that does not change. Constant is usually referred as a value.

e.g $x = 8$; $y = 10$;

here 8 and 10 are constant values.

Types of Constants :-

1) Primary constant
2) Secondary constant



1) Rules for constructing integer constant :-

a) An integer constant should be at least one digit and can not have decimal point.

b) It can be positive or negative, if sign is not given then integer constant is considered as positive.

c) NO commas or blank spaces are allowed within integer constant.

d) the range of integer constant is from -32768 to 32767

e.g. +426, 9000, -945

2) Rules for constructing float/Real constant :-

a) A float constant should be atleast one digit and must have decimal point.

b) It can be positive or negative. If sign is not given, then float constant is considered as positive.

c) No commas or blank spaces are allowed within float constant

e.g :- +780.23 , −22.56 , 970.0

3) Rules for constructing Character Constants :-

a) A single character enclosed in apostrophes is known as a character constant.

b) A character constant can be single alphabet, digit or special symbol.

c) A character constant is specified using

e.g: 'G', '8', '+'

4) Operators :- Operators are the symbols used to indicate the operations on the operands. i.e. values
e.g 10 + 20 , Here 10 and 20 are operands and + is called operator.
- An expression is consist of operators, operands.

Types of Operators :-
1) Arithmatical operator
2) Relational Operator
3) Logical operator
4) Assignment operator
5) Increment & decrement operator
6) Conditional operator
7) Bitwise operator
8) special operator

1) Arithmatical Operator :- Arithmatical operators are used to do basic operations like addition, subtraction, multiplication and division.
% is a operator called as modulus operator.
mod operator is used to find out the remainder from division.
e.g. 10 % 2 is 0 (zero).
i.e 10 is divisible by 2 so the remainder is '0' and quotient will be 5.

operator	meaning	Example	
+	addition	a+b	10+5
-	Subtraction	a-b	10-5
*	multiplication	a*b	10*5
/	division	a/b	10/5
%	modulus	a%b	10%5

2) Relational operators :— Relational operators are also known as comparison operators. These operators check the relation between two values or variables.

Operator	Meaning
<	Less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	Not equal to

3) Logical operators :— Logical operators are used to evaulate the conditions or expressions.
The logical operators are AND, OR, NOT.
The AND (&&) operator checks for all the conditions to be true then only it returns true otherwise false.
OR (||) operator checks for any one or all conditions from the given conditions to be true then it evaulates true otherwise if all conditions are false then it returns false.
NOT (!) operator checks for negative condition.

Operator	meaning
&&	AND - True if all conditions are true
||	OR— True if any one or all conditions are true
!	NOT - Negation

4) Assignment operator :-
The assignment operator is '='.
Assignment operator is used to assign some value to the variable.
e.g :- a = 10
Here we are assigning the value 10 to the variable a.

5) Increment & decrement operator :-
Increment & decrement operators are unary operators means that they take one operand to perform operation on it.
- Increment operator adds one to the variable.
- Decrement operator subtracts one from the variable.

e.g :- ++x is similar to x = x+1
if x = 5 then after ++x or x++ x will become 6.

e.g :-> --y is similar to y = y-1
or y-- is similar to y = y-1.

① ++a :- It is a pre increment operator.
Pre increment operator add one to the variable and then perform the assignment operation.

e.g : int a = 5;
int b = ++a;

O/P :- a = 6, b = 6

(2) --a :- It is a post decrement operator. Post decrement operator subtracts the one from the variable and then perform assignment operation.

e.g :- int a = 5;
int b = --a;
o/p :- a=4, b=4

(3) a++ :- post increment operator.
post increment operator do the assignment operation first and then ~~perform~~ add one to the variable.

e.g : int a = 5
int b = a++;
Here value a will be 6 and b will be 5

(4) a-- :- post decrement operator.
Post decrement operator do the assignment operation first and then subtract one from the variable.

e.g :- int a = 5;
int b = a--;
o/p :- a = 4, b = 5

preincrement ++i	post increment decrement i++
① It is known as pre-increment operator	① It is known as post increment operator
② After incrementing the value of i by one its new value is used in expression	② first the expression gets evaluated and then the value of i will be incremented by one.
③ If we use this operator in an expression then it will affect the result of that expression	③ If we use this operator in an expression then it will not affect the result of that expression

④ eg:-
int a=5;
int b = ++a;
o/p:- a=6
b=6
	④ eg:-
	int a=5;
	int b= a++;
	o/p:- a=6
	b=5

* pre decrement --i	post decrement i--
(1) It is known as pre decrement operator	(1) It is known as post decrement operator
(2) After decrementing the value of i by one its new value is used in expression	(2) first the expression gets evaluated and then the value of i will be decremented by one.
(3) If we use this operator in an expression then it will effect the result of that expression	(3) If we use this operator in an expression then it will not affects the result of that expression
(4) eg:- int a = 5; int b = --a; o/p:- a = 4 b = 4	(4) eg:- int a = 5; int b = a--; o/p:- a = 4 b = 5

(6) Conditional operator:- It is also known as Ternary operator.
Conditional operator is a combination of [? :].

Syntax:- expression 1 ? expression 2 : expression 3
It checks for the condition and evaluates the result depending on the status of condition i.e. true or false.
e.g. max = a > b ? a : b

In the above statement if a > b then the condition becomes true then max variable will store value of a. otherwise if a > b is false then max will store value of b.

WAP to find out greater number between 2 nos using conditional operator.

```
void main()
{
  int a, b, max;
  clrscr();
printf("\n enter value of a:");
scanf("%d", &a);

printf("\n enter value of b:");
scanf("%d", &b);

max = (a>b ? a : b);

Printf("\n The greater no is %d", max);
    getch();
  }
```

(7) special operator :- These operators performs special operations.
The special operators are comma, sizeof, pointer operator i.e. (&, *) and member selection operator such as →

The sizeof operator gives the size or length in the bytes of variable

program:-

```
    void main()
  {  float f;
     clrscr();
Printf("\n sizeof f is %d", sizeof(f));
    getchar();
  }
```

(8) Bitwise Operators :- Bitwise operators are used to manipulate the data at bit level. Compilers converts high level language into the low level language.
Compiler compiles the data and program in machinery code i.e. in the form of '0' and '1'. 0 and 1 are called as bits.
So to do multiple operations on these bits we require bitwise operators.

Operator	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	shift left
>>	shift right

* Operator Precedence :- Precedence means priority. One expression can include multiple operators for the purpose of calculation. According to the priorities set, the expression gets evaulate or execute. Each operator is associated with its precedence.
e.g (x == 25 + 10 && y < 10)
The highest priority is given to ( ).
The next priority is given to '+' as compare to && and relational operators. (== and <)
So that addition of 25 and 10 will be calculated.
∴ x == 35 && y < 10
now suppose x = 10 and y = 7, now < operator has got higher priority than ==, x == 35 will be tested first and then y is tested.
now, x == 35 will be become false and y < 10 will become true.

due to AND condition the combination of (false && True) will become false.

i.e. (false && true) will become false.

* Associativity:- The associativity is the feature which shows that expression evaluation of operators will take place from either left or right to left.
This indicates the direction in which evaluation takes place.

e.g: $x = 7+3-2$

- Here + and - operator has same priority and its associtivity is from left to right.
so addition will be performed first and then subtraction.
so $x = 8$.

When multiple operators in an expression has same priority then the sequence of operations to be performed is decided by their associativity.

Structure of C program:-

1) Document section:- It gives information about the program

2) Preprocessor commands:- It includes required library files.

3) Function section:- Each C program should contain exactly one main() function from where the program execution begins.

4) Declaration section:- declaration of variable will be done in declaration section.

5) Statements and expressions:- Statements and expressions are usually written in the function block. Each statement ends with semicolon (;)

6) Comments:- The comments are ignored by the compiler and it is used to add additional information about the program or statement. Comment can be written any where in the program.

eg:-

```
// Program written by Anuja  -> document section
#include<stdio.h>  -> Preprocessor command
void main()  -> function section
{
   int a;  -> declaration section
   clrscr();
printf("\n %d", a);  -> statement
   getch();
}
```

* Rules while writing the C program :—

⇒ ① Execution of every 'C' program starts with the main() function.

② Each 'C' program contains exactly one main() function.

③ Every statement in C program ends with semicolon (;)

④ Each opening brace should have its closing brace. Braces are used to define the blocks in coding.

⑤ 'C' can contain comments anywhere in the program.

⑥ The execution of program starts with the opening brace of main() function and ends with the closing brace of the main() function.

⑦ Every C program starts with #include <stdio.h>

* use of #include <stdio.h> :-
stdio.h is a standard Input output header file. stdio.h file is used to perform standard input output operations like printf() and scanf() function.

* Use of conio.h :-
conio.h is a console Input output header file. To deal with console related operations the conio.h file is included in the program. like clrscr() and getch() function.

* Use of math.h :— math.h is used to perform the mathematical operations like sqrt().

* main() function :— Every 'C' program must have main() function.
main() function is the entry point of the program execution.
It is not possible to run the program without main() function.
All the expressions and statements like declaration statement, assignment statement, arithmetic expression is usually written within the main() functions body.
main() can be written in the following forms :—

1) main()
2) int main()
3) void main()
4) main(void)
5) void main(void)
6) int main(void)

The keyword void is used which means no argument main(void) means no argument can be passed.

Void main() means main() function does not return any value.
int main() will return value to the operating system.

Syntax for main() function :—

```
main()
{
---
executable statements;
}
```

* Preprocessor directive :—

Preprocessor directive is #define.
The #define is generally used to define the constant values.
These values are generally written in upper case letter. Due to this it becomes easy to recognise the preprocessor directive.

* formatted output or printf function :—

⇒

printf() function is used for displaying a value or a data on the screen.

Syntax :—

printf ("format string", variable 1, variable 2);

* Write format string used for printf function :—

	format string	meaning	example	Result
①	%d	prints a decimal integer	int a = 5; printf ("%d", a);	5
②	%c	prints a single character	char ch = 'r' printf ("%c", ch);	r
③	%f	prints a float value	float a = 14.44; printf ("%f", a);	14.44
④	%s	prints a string	char str[10] = "abc"; printf ("%s", str);	abc

Formatted Input or scanf() function:-

⇒ scanf() function is used for accepting a value or data from keyboard.

syntax for scanf():-

scanf("format string", &variable1, &variable2);

* write format strings used for scanf() function:-

	Format string	meaning	example	Results
①	%d	Reads integer value	scanf("%d", &no);	%d accepts the number from user & it will get stored in "no" variable
②	%c	Reads a single character	scanf("%c", &ch);	%c accepts the character from user & it will get stored in "ch" variable
③	%f	Reads a float value	scanf("%f", &no);	%f accepts a float number from user and it will get stored in no variable.
④	%s	Reads a string	scanf("%s", str);	%s accepts the string from user and it will get stored in "str" variable.

* Input statements / functions :—
Input functions are used to accept the input by users for a program. User enters the data through the keyboard.

Standard input functions :—

1) scanf ( ) :— used for accepting a value or a data from keyboard.

2) getchar ( ) :— is used to get or read a one character from keyboard.

3) getch ( ) :— Accepts a character

4) gets ( ) :— It accepts and stores the sequence of characters.

* Explain getchar ( ) and putchar ( ) functions :—

⇒ getchar ( ) is used to read a one character at a time from standard input. getchar can be used to read int or char type of data.

putchar ( ) is used for the output.
It prints character argument on the screen at the current position of cursor.
It is used for int and char data types only.

example :—
```
#include <stdio.h>
int main()
{
    int a = getchar();
    putchar(a);
    return 0;
}
```

* Output statements / functions :—
Output functions are used to print the values of variables on the monitor.

Standard output functions :—

1) printf() :— It is a function used for displaying a value or data on the screen.

2) putchar() :— used to write a character on standard output / screen.

3) putch() :— It takes an ASCII int value as argument and then prints corresponding character.

4) puts() :— It takes an character array as argument and print the value stored in the character on the screen.

example :—

①
```
#include<stdio.h>
int main()
{
  char ch = getch();
  return 0;
}
```

②
```
#include<stdio.h>
int main()
{
  char ch = getch();
  putch(ch);
  return 0;
}
```

* Variable :- The entity which can be changed at different times are called as variable. These are the memory location names where the values are stored in the memory. one variable stores one value at a time.

* Variable declaration :- The variables are declared along with its data type.
Syntax for variable declaration :-
Syntax :- datatype variablename;
e.g int no;
float per;
char name[10];

* Variable initialization :- variable initialization is assigning value to for variable.
e.g int no; → variable declaration
no = 1; → variable initialization

* Declaring variable as constant :-
Constant variables are those variables whose value once assigned can not be changed throughout the program.
e.g ~~Pi~~ variable PI can be assigned a value 3.14
i.e. Const float PI = 3.14;

* Rules to declare a variable :-

1) The first character of variable name should be alphabet or underscore.
2) Except underscore no special symbols are allowed in variable name.
3) The maximum length of variable can be maximum 31 alphabets, digits or special symbol like underscore.

e.g : grade, emp_name, emp_id_no

WAP to accept amount in Rupees and convert it into paisa.

⇒
```
void main()
{
    float rupees;
    double paisa;
    clrscr();
    printf("\n Enter Rupees :");
    scanf("%f", &rupees);

    paisa = rupees * 100;

printf("\n Rupees = %f and paisa = %lf", rupees,
                                        paisa);
    getch();
}
```

Output:-

Enter Rupees: 567.76
Rupees = 576.76 paisa = 56760

* WAP to calculate simple interest

⇒
```
void main()
{
    float pamt, year, rate; interestamt;
    clrscr();
printf("\n Enter principal amount :");
scanf("%f", &pamt);
printf("\n Enter the year :");
scanf("%f", &year);
printf("\n Enter
    interestamt = (pamt * rate/100) * year;

Printf("\n The interest amount is %f",
                                    interestamt);
    getch();
```

WAP to ~~calculate~~ gross salary. Accept basic salary and calculate gross salary with 5% DA, and 15% TA on basic salary. Calculate gross salary.

```
void main()
{
  int basic, da, ta, gs;
  clrscr();
Printf("\n Enter basic salary :");
scanf("%d", &basic);

  da = (5*basic)/100;
  ta = (15*basic)/100;
  gs = basic + da + ta;

Printf("\n Basic = %d", basic);
printf("\n da = %d", da);
printf("\n ta = %d", ta);
printf("\n gs = %d", gs);
getch();
}
```

Output:-

Enter basic salary:- 5000
basic = 5000
da = 250
ta = 750
gs = 6000

WAP to ~~calculate gross salary~~. Accept basic salary and calculate gross salary with 5% DA, and 15% TA on basic salary. Calculate gross salary.

```
void main()
{
  int basic, da, ta, gs;
  clrscr();
printf("\n Enter basic salary :");
 scanf("%d", &basic);

   da = (5*basic)/100;
   ta = (15*basic)/100;
   gs = basic + da + ta;

printf("\n Basic = %d", basic);
printf("\n da = %d", da);
printf("\n ta = %d", ta);
printf("\n gs = %d", gs);
 getch();
}
```

Output:-

```
Enter basic salary:- 5000
basic = 5000
da = 250
ta = 750
gs = 6000
```

WAP to accept amount in Rupees and convert it into paisa.

⇒
```
void main()
{
   float rupees;
   double paisa;
   clrscr();
   printf("\n Enter Rupees :");
   scanf("%f", &rupees);

   paisa = rupees * 100;

printf("\n Rupees = %f and paisa = %lf", rupees,
                                          paisa);
   getch();
}
```

Output:-

Enter Rupees: 567.76
Rupees = 576.76 paisa = 56760

* WAP to calculate simple interest

⇒
```
void main()
{
   float pamt, year, rate; interestamt;
   clrscr();
printf("\n Enter principal amount :");
scanf("%f", &pamt);
printf("\n Enter the year :");
scanf("%f", &year);
printf("\n Enter
   interestamt = (pamt * rate/100) * year;

printf("\n The interest amount is %f",
                                   interestamt);
   getch();
```
}

program "Basics of C"

1) WAP to wish hello to user.

```
void main()
{
  char name[10];
    clrscr();
printf ("\n Enter name: ");
scanf ("%s", name);
printf ("\n Hello %s welcome to C ", name);
    getch();
}
```

2) WAP to find out area of circle

```
const float pi = 3.14;
void main()
{
 float rad, area;
   clrscr();
printf ("\n Enter the radius: ");
scanf ("%f", &rad);
   area = pi * rad * rad;
printf ("\n The area of circle is %f", area);
    getch();
}
```

3) WAP to convert fahrenheit temperature to celicus.

⇒

```
void main()
{
   float ftemp, celtemp;
     clrscr();
printf ("\n Enter temperature in fahrenheit: ");
scanf ("%f", &ftemp);
   celtemp = (ftemp - 32) / 1.8;
printf ("\n The temperature in celsius is %f", celtemp
    getch();
}
```

4) WAP to accept 3 nos. add them and find out average

```
void main()
{
int no1, no2, no3, total;
   float average;
   clrscr();
printf ("\n Enter three numbers :");
scanf ("%d %d %d", &no1, &no2, &no3);
   total = no1 + no2 + no3;
   average = total / 3;
printf ("\n The total is %d and average
   is %f", total, average);
   getch();
}
```

5) WAP to find out area & perimeter of rectangle.

```
void main()
{
   int len, breadth, area, perimeter;
   clrscr();
   printf ("\n Enter length and breadth :");
   scanf ("%d %d", &len, &breadth);
   area = len * breadth;
   perimeter = (2*len) + (2*breadth);

printf ("\n The area is %d", area);
printf ("\n The perimeter is %d", perimeter);
   getch();
}
```

6) WAP to find out greater of 2 nos using conditional operator.

```
void main()
{
 int no1, no2, max;
 Printf ("\n Enter the 2 nos:");
 scanf ("%d %d", &no1, &no2);
 max = (no1 > no2 ? no1 : no2);
printf ("\n The greater number is %d", max);
   getch();
}
```

7) WAP to enter basic salary. Calculate gross salary with 5% DA, 15% TA, hra=5% on basic salary. Display calculated gross salary.

```
void main()
{
 int basic, hra, da, ta, grosssalary;
 clrscr();
 Printf ("\n Enter basic salary:");
 scanf ("%d", &basic);
    da = (5 * basic)/100;
    hra = (5 * basic)/100;
    ta = (15 * basic)/100;

    grosssalary = basic + hra + da + ta;

Printf ("\n The basic salary is %d", basic);
Printf ("\n The hra is %d", hra);
Printf ("\n The da is %d", da);
Printf ("\n The ta is %d", ta);
Printf ("\n The gross salary is %d", grosssalary);
   getch();
}
```

8) WAP to display hexadecimal, decimal and octal format of the entered no.

```
void main()
{
int no;
clrscr();
printf("\n Enter the no:");
scanf("%d", &no);

printf("\n Hexadecimal no is %x", no);
printf("\n The octal no is %o", no);
printf("\n The decimal no is %d", no);
getch();
}
```

o/p:- Enter no: 15
Hexadecimal no: f
octal no: 17
decimal no: 15