Operator Precedence :-

Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence. Certain operators have higher precedence than others. For example the multiplication operator has higher precedence than addition operator.

For example $x = 7 + 3 * 2$ ∴ x is assigned 13 not 20 because * operator has higher precedence than + operators. So it first multiplied with 3*2 and then adds into 7.

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

Example :-

```
#include<stdio.h>
main()
{
int a=20, b=10, c=15, d=5, e;

e= (a+b) * c /d;
Printf ("Value of (a+b)*c/d is %d \n", e);

e =(a+b * c) /d;
Printf (" Value of ((a+b) *c) /d is= %d \n" e);

e =(a+b)*(c /d);
Printf ("Value of (a+b) *(c/d) is : %d \n", e);

e = a+(b * c) /d;
Printf ("Value of a+(b*c)/d is:%d \n" e);
}
```

Output will be :-

```
Value of (a+b)*c /d is = 90
Value of ((a+b) *c /d is= 90
Value of (a+b)*(c /d) is =90
Value of a+(b*c)/d is =50
```

Calculate the following Expression in C Language precendence

1) 100 + 200/10 - 3*10
100 + 200/10 - 30
100 + 20 - 30
= 120 - 30

2) Program in C of this expression in C is :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a = 100, b = 200 c = 10 d = 3 e = 10
Heir;
Heir = a + b/c - d*e
Printf("\n output is = %d", Heir);
getch();
}
```

3) 250/50 + 300 * 2 + 450%70 - 35
250/50 + 600 + 450%70 - 35
= 5 + 600 + 450%70 - 35
= 5 + 600 + 30 - 35
= 605 - 5
= 600

Program for this Expression is :-

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

```
int a=250, b=50, c=3000, d=2, e=300, d=2, e=450
f=70, g=35, pro;
clrscr();
pro = a/6 + c*d + e%f - g;
printf("\n output is = %d", pro);
getch();
}
```

3.) $5+2*10/7+13\%5+19/2-5*2-1$

$=5+20/7+13\%5+19/2-10-1$

$=5+2+3+9-11$

$=7+3-2$

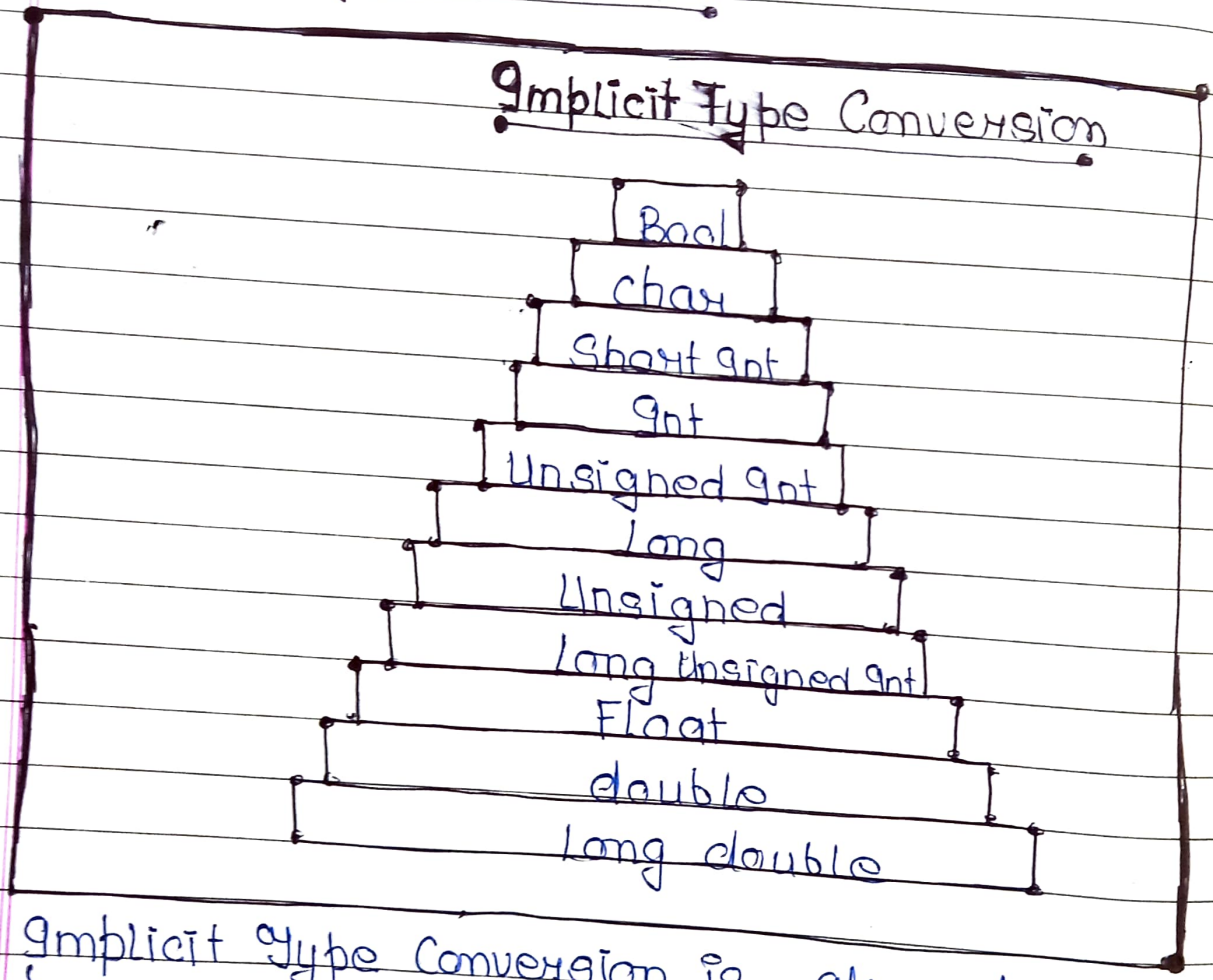
$=10-2=8$

Program for expression is :-

```
#include<stdio.h>
Void main()
{
int a=5, b=2, c=10, d=7, e=13, f=5, g=19, h=2, i=5, j=2;
k=1, hier;
clrscr();
hier = a+b*c/d+e%f+g/h-i*j-k;
printf("\n output of expression is = %d", hier);
getch();
}
```

Type Conversion in C :- A type cast is basically a conversion from one type to another. There are two types of type conversion.

1. Implicit Type Conversion :-



Implicit type Conversion is also known as 'automatic type Conversion'. Implicit type Conversion is done by the compiler on its own, without any external trigger from the User. This conversion generally takes place when in an expression more than one data type is present. In such condition type conversion takes place to avoid loss of data. All the data types of the variable are upgraded to the data type of the

Variable with largest data type.

Bool → char → short int → int → unsigned int → long → unsigned long long → float → double → long double

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

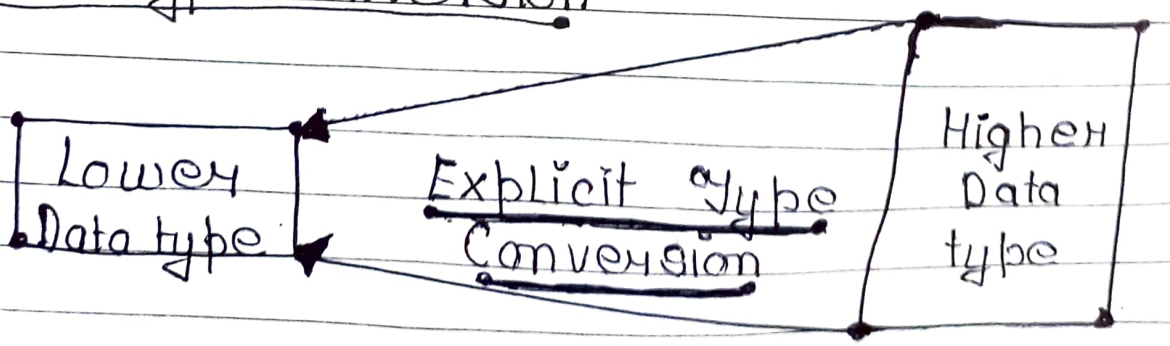
Example of Implicit type Conversion

```
// Example of Implicit type Conversion
#include <stdio.h>
Void main ()
{
int x = 10;
char y = 'a';
x = x+y;
Float z = x+1.0
Printf ("x =%d, Z = %f", x, z);
}
```

Output:

X = 107, Z = 108.000000

2. Explicit type Conversion



This process is also called type Casting and it is user defined - Here the user can type Cast the result to make it of particular data type

The syntax in C is

(type) expression

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

```
#include<stdio.h>
void main()
{
double x=1.2;
int sum=(int)x+1;
printf("sum=%d",sum);
}
```

Output:-

Sum = 2

Advantages of type Conversion :-

1) This is done to take advantage of certain features of type hierarchies or type representations.

2) It helps us to compute expressions containing variables of different data types.

Chapter - 4

Input and Output operations

4.1 Console Input/output in C Programming :-

In order to keep C programming language compact Dennis Ritchie removed anything related to input or output from the definition of the language. Therefore C has no provisions for input and output of data from input and output devices. In order to solve this little discrepancy, C developers developed several standard input and output functions and placed them in C libraries. All these libraries are accessible by all C compilers

Types of Input/output functions :- A lot of input/output functions have been defined in standard libraries. These can be classified as follows

1. **Console I/O functions :-** These functions allows us to receive input from input devices like keyboard and provide output to output devices like visual display unit.

2. **File I/O functions :-** These functions allow us to access the hard disk or floppy disk to perform input and output.

Console I/O functions :- A Console Comprises the VDS and the keyboard the Console Input and output functions can be classified into two categories

1:- Formatted Console Input functions :-

1:1 Scanf() :- Scanf() is the formatted Console Input function which reads formatted Input from stdin(Standard Input). It can read any integer, float, character string etc data from the user. The Syntax of using Scanf() is as follows:

Syntax

Scanf("Format Specifier", arguments address);

Example :-

Scanf("%d", &Sum);

In this example %d is format Specifier for an integer. &num indicates the address of num where value is to be stored under the Variable name 'num'.

Disadvantages :-

One disadvantage of scanf() when taking string Input is that it will ignore the string that has been entered after a blank space. Thus, Scanf() does not take multi-word string Inputs. To take multi word string Input we should use gets() method.

2.4 Printf() :- Printf() is the formatted Console output function which prints the formatted output to the stdout (Standard output) It can display integers, floating point values, characters, string etc as indicated by the User. The Syntax of using printf is as follows

```
printf("text");
```

This shall simply print "text" on output screen. In order to print any variable value along with text we have the following format

```
printf("text <format Specifier>", variable);
```

This shall print the value of the variable in format specified by format Specifier
For example, `printf("marks: %d", marks);`
This has the format specifier as %d which stand for integer data and it will print the marks in integer format.

Limitation :- For Floating point values the format Specifier is %f and it will print 6 decimal places after the floating point. For example, 3.6 will be displayed as 3.600000. In order to limit the decimal places after floating point, the format specifier can be written as %0.3f for 3 decimal places after floating point

2.5 Unformatted Console I/O functions:- The Unformatted Console input/output functions deal with a single character or a string of characters. Let us see how all

these Unformatted functions work

1. getch() : getch() function is also an unformatted input function supported by C Language. The function reads a character from the keyboard and does not echoes it to the screen. It is present in the header file "Conio.h". This function returns the character that was typed last or was typed most recently.

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main()
 {
Printf("Your name is Shubham");
getch()
 }
```

2. getche() : getche() is an unformatted input function supported by C language. It is an unbuffered input dev function. This function waits for user to press a key from the keyboard and instantly transfers the value into variable. This function is same as getch(). The only difference is that it displays the most recently used character. It is also present in <conio.h>

Its difference from getchar() function is that it does not requires pressing of enter key after typing the character.

Syntax :-

Variable Name = getche();

Example: char rec;
rec = getche();

On executing above statement, the key pressed by user will be assigned to variable "rec".

3. getchar() :- getchar is an Unformatted Input function supported by C Language. This is buffered input output function. This function is used to get or read a single character from the standard input device. The definition of this function is stored in header file "stdio.h" getchar() is a macro that works in a similar manner as getch() and also displays(echoes) the character but it needs the user to press the enter key after the character.

Syntax :-

Variable Name = getchar();
[Here variable name should be valid variable name in C, conforming to the variable naming conventions. When you execute the program, the the function waits for the user to press a key

Example :-

```
char rec;
rec = getchar();
```

On executing above statement, the key pressed by user will be assigned to the variable "rec".

4. putchar() :- Putchar() is a single character output function. It just transmits a single character to standard output device (monitor). The function takes the name of variable as argument and displays the character stored in variable on screen. The definition of this function is stored in header file "stdio.h"

Syntax :-

```
putchar(variable name);
```

Example :-

```
char c;
c = 'A';
putchar(c);
```

5. gets() :- The gets() function is used to input strings from the keyboard. The advantage of this function is that it can store blank spaces as a part of string. In this way, it is better than scanf() which fails to do the same. It takes

a string input (single word or multi-word) from user - it terminates when enter key is pressed

Syntax :-

gets(string name);

5. puts :- The puts() function is used to print strings on the standard display device the advantage using the function is that it automatically puts a new line character in front of the string being displayed so output always appears in new line. It is used to print string to a console.

Syntax :-

puts(string name);

Format Specifier for I/O Here's a list of Commonly used C data types and their format specifier

	Data type	Format Specifier
1.	Int	%d, %i
2.	Char	%c
3.	Float	%f
4.	double	%lf
5.	Short int	%hd
6.	Unsigned int	%u

Long int	%ld
Long Unsigned int	%lu
Long Long int	%llu
Unsigned char	%c

Program to Illustrate Putchar() function :-

```
// Program to demonstrate the use of Putchar()
#include <stdio.h>
main()
{
char ch;
Printf("\n Enter your single character");
ch = getchar();
printf("\n You entered the character ->");
putchar(ch);
}
```

Program in C to demonstrate the use of getch();, gets, puts in C Language :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
char ch[10];
clrscr();
Printf("Enter your name");
gets(ch);
```

```
printf("\n Your name is :");
puts(ch);
getch();
}
```