C

C is computer programming language developed in 1972 by Dennis M. Ritchie.

C programming language developed in Bell Laboratories to develop the UNIX operating system.

C is a high-level language which can be easily read and written by any one.

C is a **case-sensitive** programming language. Thus, "HELLO" and "Hello" two different identifiers in C.

A simple C program

A simple C program basically consists of the following parts −

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Basic structure of C program

```
#include <stdio.h>

void main()
{


getch();

}
```

Above shown the basic structure of C program. For every program we have to write above mention basic structure.

How to write or print

We can write or print in C language by using **printf()** function. **printf()** function is a predefined function. for example: code to write “hello” is as follows:

printf(“hello”);

output: hello

Note: anything written within double code in a printf() function will be shown in console.

A simple C program to print “Hello”

```
#include <stdio.h>

#include<conio.h>

void main()
{

  /*  first C program */

  printf("Hello");

  getch();

}
```

Output: Hello

Description

- First line *#include <stdio.h>* is a preprocessor, which instruct to C compiler to include stdio.h file. stdio.h (standard input output) is used for printf function.extension of header file is .h.
- In second line *#include <conio.h>*, conio.h header used in C programming contains functions for console input/output. Some of the most commonly used functions of conio.h are clrscr, getch, getche, kbhit etc.

- *void main()* is the main function from where the program execution begins.
- /*.....*/ is a "comments" in the program. This comments will be ignored by the compiler during execution of program
- *printf()* is a predefined function in C present in stdio.h header file which is used to print message.

Compile and Execute C Program

How to save and compile the source code and run it:

- Write the above-mentioned code in Terbo C.
- Save the file as *hello.c*
- To compile your code: by pressing Alt+f9
- To execute your program, press Ctrl+f9.
- Output *"Hello" will* print on the screen.

Example: Write a program to write three times "Hello".

```
#include <stdio.h>
#include<conio.h>
void main()
{
   printf("Hello Hello Hello");
}
```

Output: Hello Hello Hello

```
#include <stdio.h>
#include<conio.h>
void main()
{
        printf("Hello");
        printf("Hello");
        printf("Hello");
}
```

Output: Hello Hello Hello

Example: Write a program to write “Hello” in three lines.

```
#include <stdio.h>
#include<conio.h>
void main()
{
        printf("Hello");
        printf("\n Hello");      //   \n is used for new line
        printf("\n Hello");
}
```

Output: Hello
Hello
Hello

Data Type

There are four types of data type in C language. They are as follows:

Types	Data Types
Basic data types	int, char, float, double
Enumeration data type	enum
Derived data type	pointer, array, structure, union
Void data type	void

Basic Data Types in C Language

Integer Data Type

Integer data (example 1,2,3,4,5,6, ….. etc.) is stored in int ,short and long data type.

Type	Storage size	Value range
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Example: Addition of integer number.

```
#include<stdio.h>
void  main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("Sum of x+y = %i", z);

}
```

Output: Sum of x+y =35

Note: Here %i is a format specifier which is used to print integer value.

USE OF " %d "format specifier

```
#include<stdio.h>
void main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("Sum of x+y = %d", z);

}
```

Output: Sum of x+y =35

Question: Find out output.

```
#include<stdio.h>
void  main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("value of x, y & z = %d%d%d", x,y, z);

}
```

Output: value of x, y & z = 10 25 35

Question: Find out output.

```
#include<stdio.h>
void main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("value of x, y & z = %d", x,y, z);

}
```

Output: value of x, y & z = 10

Question: Find out output.

```
#include<stdio.h>
void main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("value of x, y & z = %d%i%i", x,y, z);

}
```

Output: **value of x, y & z = 10 20 30**

Question: Find out output.

```
#include<stdio.h>
void main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        int z=x+y;      // add the values of variable x & y and store in variable z
        printf("value of x, y & z = %.3f%.3f%.3f", x,y, z);

}
```

Output: **value of x, y & z = 10.000 25.000 35.000**

Question: Find out output.

```
#include<stdio.h>
void main()
{       int x=10;               //  initialize integer variable x to 10
        int y=25;               //  initialize integer variable y to 25

        float  z=x+y;   // add the values of variable x & y and store in variable z
        printf("value of z = %f", z);

}
```

Output: **value of z = 35.000**

Character Data Type

Character data (example ‘a’ , ‘b’ , ‘f’, ….. etc.) is stored in “char” data type.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127

Example: Declare and initialize integer variable.

```
#include<stdio.h>
void main()
{       char c1= 'A';           //  initialize character variable c1 to A
        char c2 =66;            // ASCII value of 66 is B, in c2 character B will store

        printf("value of variable c1 &c2 = %c%c", c1,c2);

}
```

Output
value of variable c1 &c2 =A B

Question: Find out output.

```
#include<stdio.h>
void main()
{       char c1= ‘A’;           //  initialize character variable c1 to A
        char c2 =’B’;           // initialize character variable c2 to B

        printf("value of variable c1 &c2 = %d%i", c1,c2);
```

}

Output
value of variable c1 &c2 =65 66

Float Data Type

Floating point value (example 11.23, 333.3330 etc.) can be stored in data type float and double. Small floating value can be stored in float and large floating stored in double.

Type	Storage size	Value range	Precision
float	4 byte	3.4E-38 to 3.4E+38	6 decimal places
double	8 byte	1.7E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 3.4E+4932	19 decimal places

Example: Declare and initialize float & double variable.

```
#include<stdio.h>
int main()
{       float x=10.00;          //  initialize float variable x to 10.00
        float y=25.23;          //  initialize float variable y to 25.23

        float z=x+y;    // add the values of variable x & y and store in variable z

        printf("Sum of x+y = %f", z);
```

```
    double d = 11.234;      // initialize double variable d to 11.234
    printf("\n d= %f", d);          // \n is used for new line
}
```

Output:

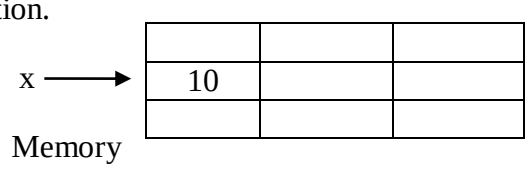
Sum of x+y = 35.230000

d= 11.234000

Variables

A variable is a name given to memory location.

int x =10;



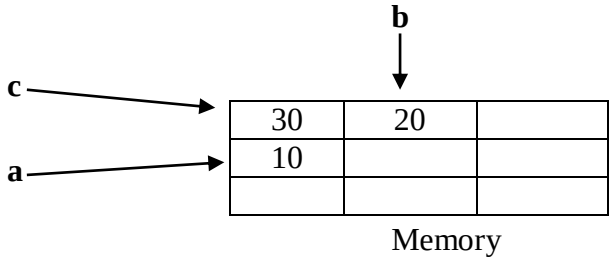
here, x is a integer variable.

Integer variable x is a name given to memory location and where we stored integer value10.

Initialize multiple variable : we can also initialize multiple by comma separated list.

int a,b,c;

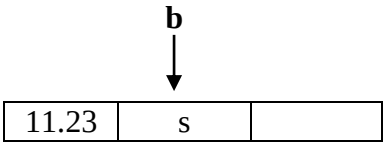
a =10;
b = 20;
c =30;



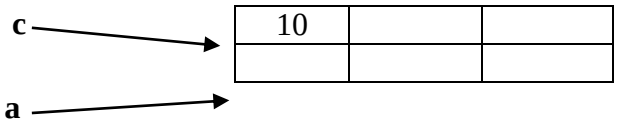
Here, we have initialized three integer variable a,b & c. variable 'a' stores 10, variable 'b' stores 20 and variable 'c' stores 30.

Initialize multiple variable : we can also initialize multiple by comma separated list.

int a;
char b;
float c;



a =10;
b = ‘S’;
c =11.23;



Memory

Format Specifiers in C

In C programming there are number of data types and format specifiers is used to defines the type of data to be printed. Whether to print output or to take input in both case we required format specifiers. Format specifiers are also called as format string.

list of format specifiers used in C language.

Format specifier	Description	Supported data types
%c	Character	char unsigned char
%d	Signed Integer	short unsigned short int long
%e or %E	Scientific notation of float values(exponent form)	float double
%f	Floating point	float
%g or %G	Floating point	float double
%hi	Signed Integer(Short)	short
%hu	Unsigned Integer(Short)	unsigned short

Format specifier	Description	Supported data types
%i	Signed Integer	short unsigned short int long
%l or %ld or %li	Signed Integer	long
%lf	Floating point	double
%Lf	Floating point	long double
%lu	Unsigned integer	unsigned int unsigned long
%s	String	char *
%u	Unsigned Integer	unsigned int unsigned long

How to print integer character float and double using format string

Print integer : we can print integer value by using %d. for example:

```
int i = 10;

printf(" value of i  =  %d"  , i );
```

Output: value of i = 10

Here whatever written within double code " " print as it is. %d is used to tell compiler print integer value. And value of " i (integer value) " will be print.

Print multiple integer : suppose we want to print two integer value then we need to write two times %d. for example:

```
int i = 10;
int j=20;
```

printf(“ value of i & j = %d%d” , i,j);

Output: value of i & j = 10 20

Here whatever written within double code “ ” print as it is. %d%d is used to tell compiler print two integer value. And value of “ i & j (integer value) ” will be print.

Print Character and float value : suppose we want to print two character and float value then we need two format string %c and %f respectively. for example:

```
char c = ‘A’;
float f=20.11;

printf(“ value of c & f = %c%f” , c,f );
```

Output: value of c & j = A 20.1100

Here whatever written within double code “ ” print as it is. %d%d is used to tell compiler print two integer value. And value of “ i & j (integer value) ” will be print.

Escape sequences

An escape sequence is a sequence of characters that does not represent itself when used inside a character or string literal, but is translated into another character or a sequence of characters that may be difficult or impossible to represent directly.

In C, all escape sequences consist of two or more characters, the first of which is the backslash, `\` (called the "Escape character"); the remaining characters determine the interpretation of the escape sequence. For example, `\n` is an escape sequence that denotes a newline character.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab

\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

```
1. #include<stdio.h>
2. int main(){
3.     int number=50;
4.     printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
5. return 0;
6. }
```

Output:

```
You
are
learning
'c' language
"Do you know C language"
```

How to take input from user : scanf()

scanf() function is used to read character, string, numeric data from keyboard. The **scanf() function allows you to accept input from** keyboard.

Syntax

scanf(char *format, arg1, arg2, …)

This function take input using standard input (keyboard) and store it in variable accordingly. It returns the number of items successfully read. Formal parameter arg1, agr2, .. must be a pointer

Take integer input from user : we can take input integer value by user using %d format string. for example:

int i ;

scanf(“ %d” , &i);

above line will read an integer value that the user enters on the keyboard. %d is used to tell compiler to read integers value from keyboard and store in a integer variable i.

Example: WAP to take two numbers from user and perform addition of two integer number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c;
                printf("Enter 1st number in a= \n");
                scanf("%d",&a);                 // 2 enter by user
                printf("Enter 2nd number in b= \n");
                scanf("%d",&b);                 // 4 enter by user
        c=a+b;
        printf("value of a,b & c= %d,%d,%d", a,b,c);

        getch();
}
```

Output: Enter 1st number in a=
2 // 2 enter by user

```
Enter 2nd number in b=
4                              // 4 enter by user
value of a,b & c= 2,4,6
```

Take multiple integer input from user: we can take two integer value as s input by user using two times %d format string. for example:

```
int i ,j;

scanf(" %d%d"  ,& i, &j );
```

above line will read two integer value that the user enters on the keyboard. Two times %d is used to tell compiler to read two integers value from keyboard and store in a integer variable i & j

Note: similarly to take char, float value from user %c & %f formatted string used in a scanf() function.

Example: WAP to take two numbers from user and perform addition of two integer number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
printf("Enter two number = \n");
scanf("%d%d",&a,&b);                    // 2 and 4 enter by user
c=a+b;
printf("value of a,b & c= %d,%d,%d", a,b,c);
getch();
}
```

```
Output:    Enter two number in a=
           2                            // 2 enter by user
           4                            // 4 enter by user
           value of a, b & c= 2,4,6
```

Tokens in C

In a C language, token is either a keyword, a symbol, an identifier, a constant or a string literal. A C program consists of various tokens for example, the following C program consists of five tokens −

```
printf("Hello, World! \n");
```

The individual tokens are −

```
printf
(
"Hello"
)
;
```

Comments

Comments are the text in C program and they are ignored by the compiler during compilation and execution.

Single line comment

// first C program

Multiline comments

They start with /* and terminate with the characters */ as shown below −

```
/* first C program  */
```

Anything written within Comment will never execute. You can’t write comments within comments.

Identifiers

In a C language, identifier is a name given to any variable, function, or any other user-defined item. C language identifier starts with a letter A to Z, a to z, digits (0 to 9) or an underscore '_' followed by zero.

C does not allow punctuation characters such as @, $, and % within identifiers. Here are some examples of acceptable identifiers −

```
kumar     aditya    abc   move_name   a_123
name70        j     a23b9       myDemo
```

Keywords

In a C language keywords are the reserved words whose meaning already explained in a compiler. These reserved words can't be used as a variables or identifier names. There are **32**Keywords in C

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

ASCII(American Standard Code for Information Interchange)

It is a 7 bits alphanumeric code.

128 character can be represented by ASCII.

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Operators in a C Language

An operator is a symbol. It is used to tells the compiler to perform specific mathematical or logical functions. C language have a following types of built-in operators:−

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

Arithmetic Operators

C language have a following arithmetic operators. Assume variable **i** holds 100 and variable **j** holds 200 then

Operator	Description	Example
+	Adds two operands.	i + j = 300
−	Subtracts second operand from the first.	i − j = -100
*	Multiplies two operands.	i * j = 20000
/	Divide.	j / i = 2
%	Modulus Operator returns the remainder of integer division.	j % i = 0
++	Increment operator increases the value of variable by one.	i++ = 101
--	Decrement operator decreases the value of variable by one.	i-- = 99

Example : Write a program to demonstrate the various arithmetic operators.

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

```
    int a=10;
    int b=25;
    int c,d ,e,f;

    c=a+b;          // add value of variable a&b and store in variable c (c=35)
    d=b-a;          // subtract value of variable a from b and store in variable d (d=15)
    e=b/a;          // e=2
    f=b%a;          //f=5

    printf(" value of c= %i", c);
    printf("\n value of  d= %i", d);
    printf("\n  value of  e= %i", e);
    printf("\n value of  f= %i", f);
}
```

Output

```
value of  c= 35
value of  d= 15
value of  e= 2
value of  f= 5
```

Increment and Decrement Operator

1. Increment operator ++
2. Decrement operator --

In a C language, Increment operator (++) increases the value of variable by one and Decrement operator (--) decreases the value of variable by one.

There are two form of increment and decrement operator:

1. Prefix form
2. Postfix form

Prefix Form : In the prefix expression operator appears in the expression before the operands.

Example : ++A

In the prefix form first the value of operand is increment or decrement than the value of operand is used in expression.

For Example: **int i = 10;**
j = ++i ; // j=11 , i=11

Postfix Form: In the postfix expression operator appears in the expression after the operands. Example : Example : A++

In the postfix form first the value of operand is used in expression than value of operand is incremented or decremented.

For Example: **int i = 10;**
j = i++ ; // j=10, i=11

Example 1: Write a program to demonstrate the increment and decrement operator.

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

        b=++a;                  // b=11, a=11
        c=a++;                  //c= 11, a=12

        printf(" value of a= %i", a);
        printf("\n value of  b= %i", b);
        printf("\n  value of  c= %i", c);

  }
```

Output:

```
value of a= 12
value of  b= 11
value of  c= 11
```

Example 2: Write a program to demonstrate the increment and decrement operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c,d,e;
        a=3;
```

```
        b=2;
            c=++b;
            d=b*a++;
            e=b++ * c++;
        printf("value of a,b ,c , d & e = %d,%d,%d,%d,%d", a,b,c,d,e);
getch();
}
```

Output: value of a,b ,c , d & e = 4, 4, 4, 9, 9

Relational Operators

C Language has a following relational operators. For example: Assume variable **A** holds 100 and variable **B** holds 200 then −

Operator	Description	Example
==	Equals to	(A == B) is not true.
!=	Not equal	(A != B) is true.
>	Greater than	(A > B) is not true.
<	Less than	(A < B) is true.
>=	Greater than or equals to	(A >= B) is not true.
<=	Less than or equals to	(A <= B) is true.

Example: Write a C Program to find the largest of two numbers.

```
#include<stdio.h>
void main()
{
        int a, b, big;
        a = 10;
        b=20;

        if(a>b)                 // if condition is true move inside if-statement
        {       big=a;

        }
        else
        {       big=b;

        }
        printf("Biggest of the two number is = %d",big);
}
```

Output: Biggest of the two number is = 20

Logical Operators

There are three logical operators in C language. Assume variable **A** and B holds 1 and 0 respectively then –

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

!	NOT Operator (this operator reverse the logical state of operand. For example: If condition is true, then NOT operator will make it false)	!(A && B) is true.

Bitwise Operators

In a C programming language Bitwise operator works on bits and perform bit-by-bit operation. There are following bitwise operators in C:-

Operator	Description
&	Binary AND Operator
|	Binary OR Operator
^	Binary XOR Operator
~	Binary One's Complement Operator is unary
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator

The truth tables for &, |, and ^ is as follows –

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

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

A in binary = 0011 1100 (60)

B in binary = 0000 1101 (13)

Bitwise AND (&)

A = 0011 1100 (60)

& B = 0000 1101 (13)

0000 1100 (12)

Bitwise OR (|)

A = 0011 1100 (60)

| B = 0000 1101 (13)

0011 1101 (61)

Bitwise XOR (|)

A = 0011 1100 (60)

^ B = 0000 1101 (13)

0011 0001 (49)

Binary Left Shift Operator

Left shift operator shift all the bits in a left direction to specified number of times.

Exapmle:

```
int i = 16;              // 0001 0000
i = i<<2;                // 0100 0000  (shift left 2 bits, front 2 bits will be lost, and append 2 zero)
printf("i = %d", i);
```

Output:

i = 64

Binary Right Shift Operator

Left shift operator shift all the bits in a left direction to specified number of times.

Exapmle:

```
int i = 16;              // 0001 0000
i = i>>2;                // 0000 0100  (shift right 2 bits, last 2 bits will be lost, and 2 zero add in front)
```

printf(“i = %d”, i);

Output:

i = 4

Example: Write a program to demonstrate Bitwise operator.

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

        int a=60;
        int b=13;
        int c,d,e,f,g,h;

        c = ~a;                 //c = -61
        d= a|b;                 //d= 61
        e= a&b;                 // e= 12
        f= a^b;                 //f=  49

        printf("\n value of c = %d", c);
        printf("\n value of d = %d", d);
        printf("\n value of e = %d", e);
        printf("\n value of f = %d", f);

        g=h= 16;
        g = g<<2;               //g= 64
        h=h>>2;                 // h=4;

        printf("\n value of g = %d", g);
        printf("\n value of h = %d", h);

  }
```

Output:

value of c = -61
value of d = 61
value of e = 12
value of f = 49

value of g = 64
value of h = 4

Assignment Operators

In a C language there are following assignment operators −

Operator	Description
=	assignment operator (it assigns values of right side operands to left side operand)
+=	Add assignment operator
-=	Subtract assignment operator
*=	Multiply assignment operator
/=	Divide assignment operator
%=	Modulus assignment operator
<<=	Left shift assignment operator
>>=	Right shift assignment operator
&=	Bitwise AND assignment operator
^=	Bitwise exclusive OR assignment operator

|=	Bitwise inclusive OR assignment operator

Add Assignment Operator

Example:

```
int a = 5;        // now set the value of a = 5;
a+=4;             // add assignment operator add 4 with a's value(5) and save new value(9) again in a.
 printf("\n value of a = %d", a);
```

Output:

value of a = 9

Subtract Assignment Operator

Example:

```
int a = 15;       // now set the value of a = 5;
a-=4;             // subtract assignment operator subtract 4 from a's value(15) and save new value(11) again in a.
 printf("\n value of a = %d", a);
```

Output:

value of a = 11

Multiply assignment operator

Example:

```
int a = 5;        // now set the value of a = 5;
a*=4;             // multiply assignment operator, multiply 4 with  a's value(5) and save new value(20) again in a.
 printf("\n value of a = %d", a);
```

Output:

value of a = 20

Precedence of operators

If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.

In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).

Example of precedence

```
(1 > 2 + 3 && 4)

This expression is equivalent to:

((1 > (2 + 3)) && 4)

i.e, (2 + 3) executes first resulting into 5

then, first part of the expression (1 > 5) executes resulting into 0 (false)

then, (0 && 4) executes resulting into 0 (false)
```

Output

```
0
```

Associativity of operators

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

Example of associativity

```
1 == 2 != 3
```

Here, operators `==` and `!=` have same precedence. The associativity of both `==`and `!=` is left to right, i.e, the expression on the left is executed first and moves towards the right.

Thus, the expression above is equivalent to :

```
((1 == 2) != 3)

i.e, (1 == 2) executes first resulting into 0 (false)

then, (0 != 3) executes resulting into 1 (true)
```

Output

```
1
```

The table below shows all the operators in C with precedence and associativity.

Note: Precedence of operators decreases from top to bottom in the given table.

Summary of C operators with precedence and associativity

Operator	Meaning of operator	Associativity
() [] -> .	Functional call Array element reference Indirect member selection Direct member selection	Left to right

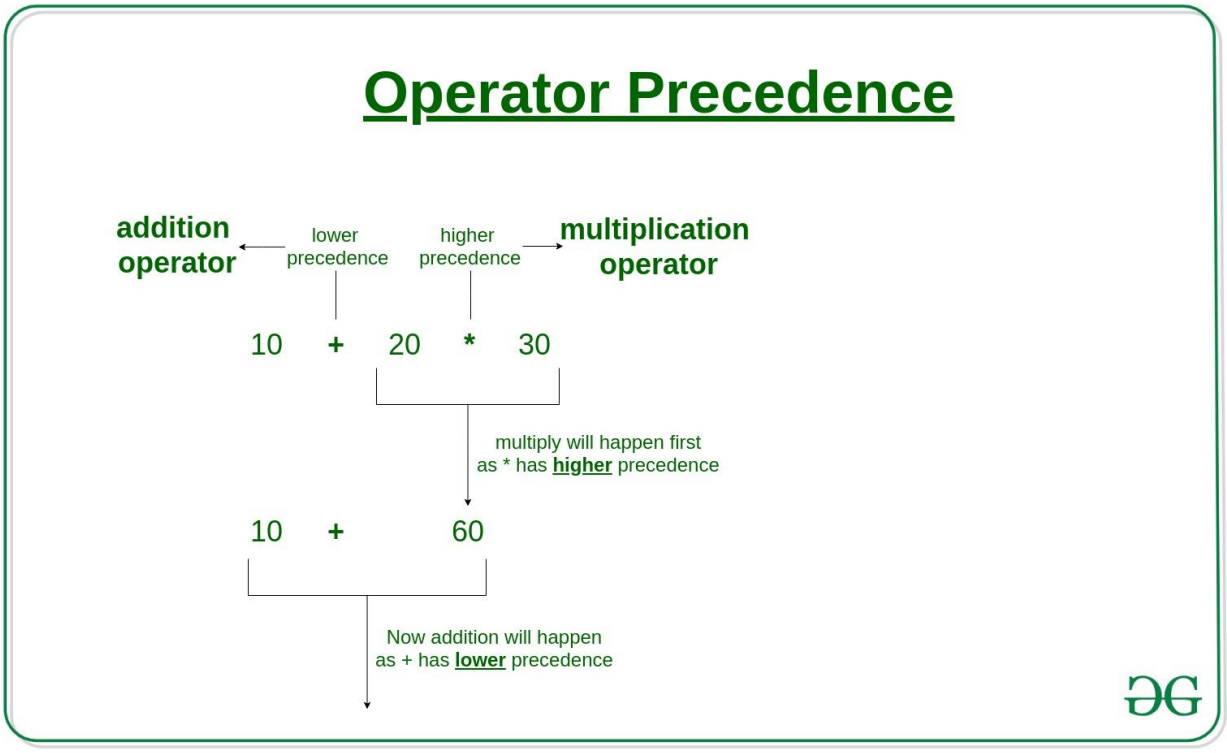
Operator	Meaning of operator	Associativity
! ~ + - ++ -- & * sizeof (type)	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference Operator(Address) Pointer reference Returns the size of an object Type cast(conversion)	Right to left
* / %	Multiply Divide Remainder	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
<< >>	Left shift Right shift	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right
== !=	Equal to Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right

Operator	Meaning of operator	Associativity
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional Operator	Right to left
=	Simple assignment	Right to left
*=	Assign product	
/=	Assign quotient	
%=	Assign remainder	
-=	Assign sum	
&=	Assign difference	
^=	Assign bitwise AND	
|=	Assign bitwise XOR	
<<=	Assign bitwise OR	
>>=	Assign left shift	
,	Assign right shift	
	Separator of expressions	Left to right

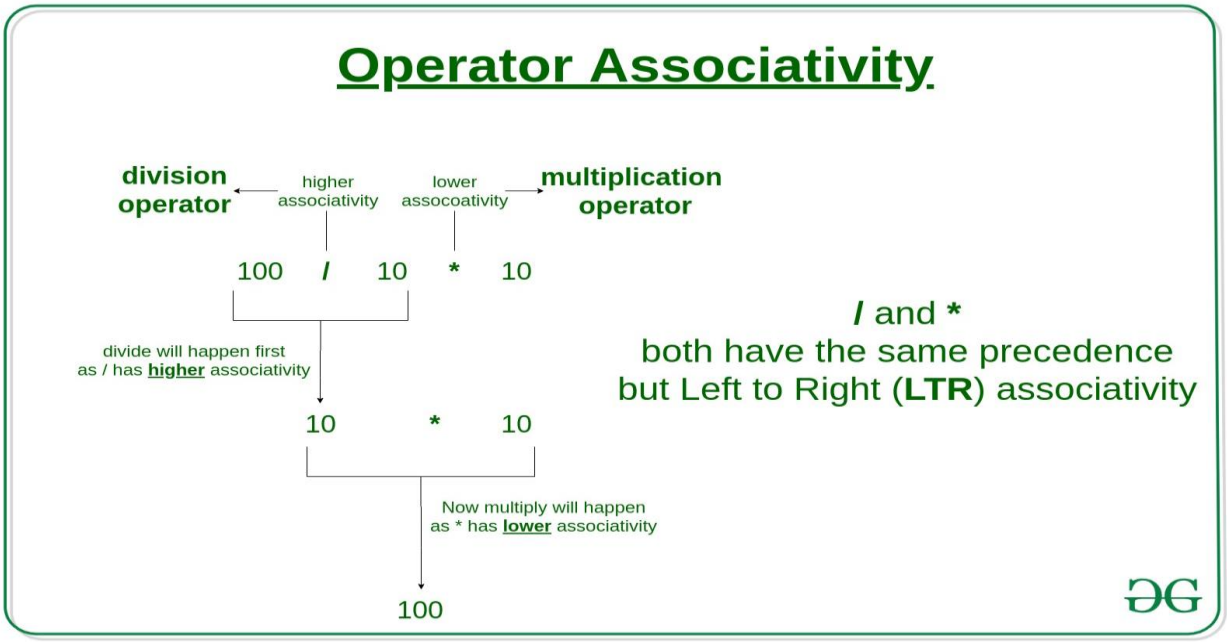
For example: Solve

```
10 + 20 * 30
```

10 + 20 * 30 is calculated as **10 + (20 * 30)**
and not as **(10 + 20) * 30**

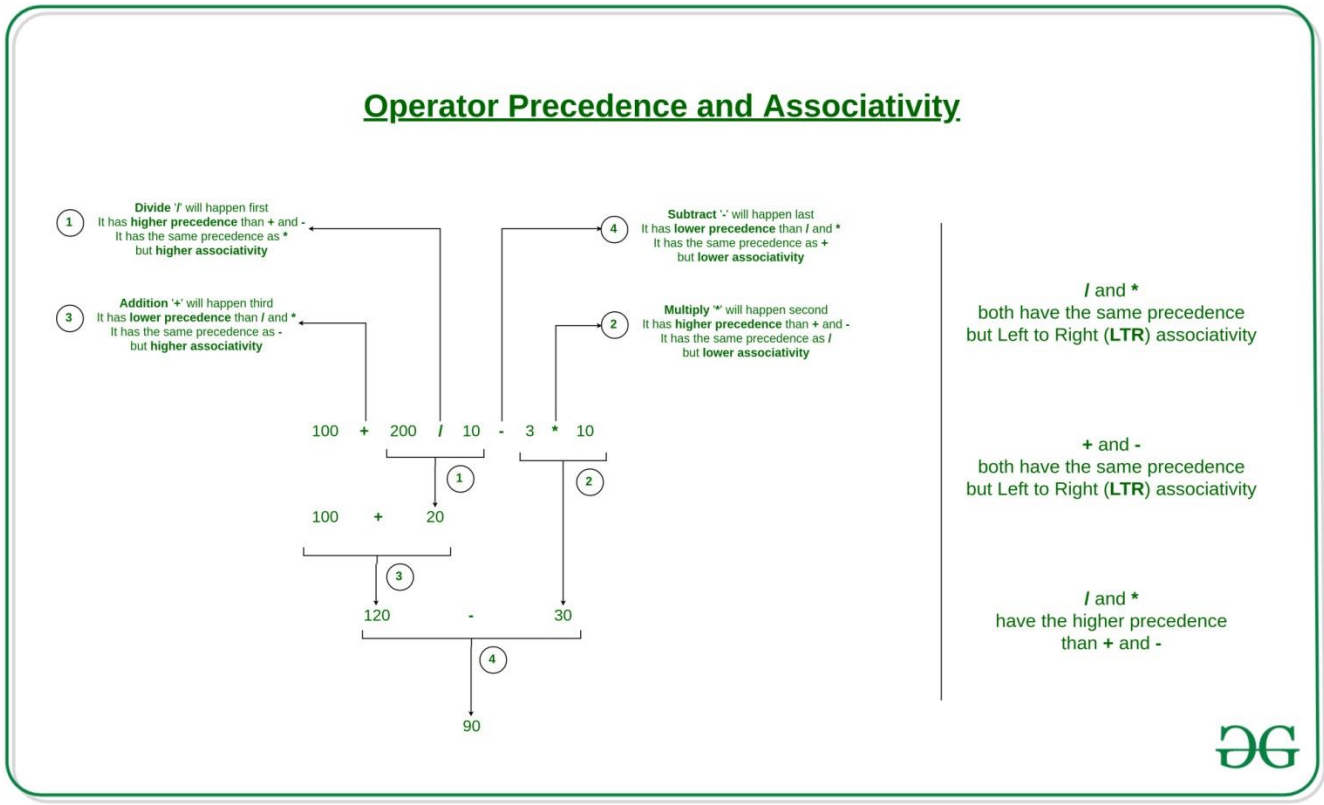


Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either**L**eft **t**o **R**ight or **R**ight **t**o **L**eft.
For example: '*' and '/' have same precedence and their associativity is **L**eft **t**o **R**ight, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".

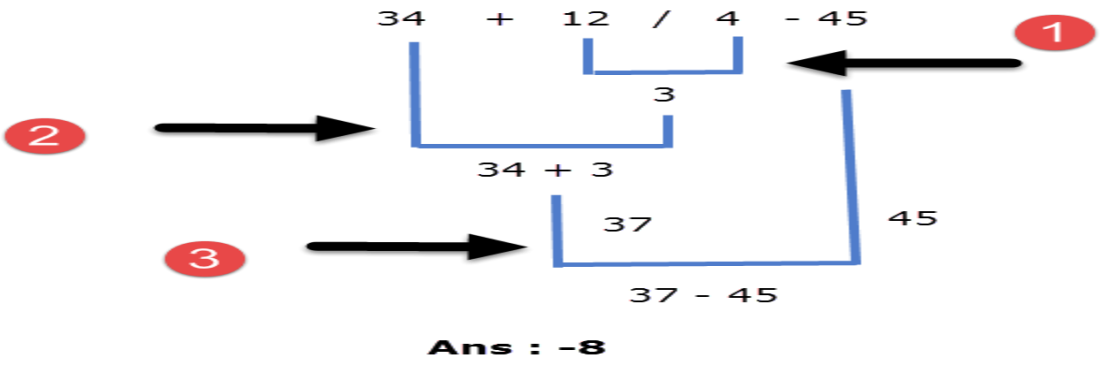


For example: Solve

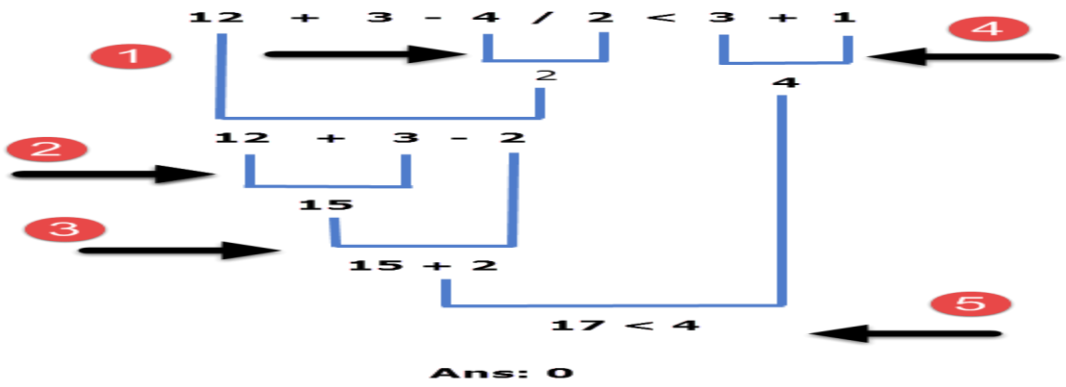
```
100 + 200 / 10 - 3 * 10
```



For example: Solve
34 + 12/4 – 45



For example: Solve 12 + 3 - 4 / 2 < 3 + 1



Example

Try the following example to understand operator precedence in C −

Live Demo

```
#include <stdio.h>

main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;      // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;    // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;     //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

When you compile and execute the above program, it produces the following result −

```
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
```

What is the output of below program

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

  c = a * 2 + b/2;

  printf("\n output = %d", c);
 }
```

Output: output =15

Syntax errors

Errors that occur when you **violate the rules**of writing C/C++ syntax are known as syntax errors.
A syntax error is an error in the source code of a program.
This error find out at the time of compilation means it is a compile time error.
This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by compiler and thus are known as compile-time errors. Most frequent syntax errors are:

- Missing Parenthesis (**}**)
- Printing the value of variable without declaring it
- Spelling mistakes
- Missing out quotes
- Missing out brackets
- Using upper case characters in key words e.g. IF instead of if
- Missing out a colon or semicolon at end of a statement
- Using tokens in the wrong order

Missing semicolon

```
#include<stdio.h>
void main()
{
   int x = 10;
   int y = 15;

   printf("%d%d", (x, y)) // semicolon missed
}
```

Output: Compile time error: // syntax error , Missing semicolon **(}**)

Missing Parenthesis (})

```
#include<stdio.h>
void main()
                                        // Parenthesis ({) missed
   int x = 10;
   int y = 15;

   printf("%d%d", (x, y));
}
```

Output: Compile time error: // syntax error , Missing Parenthesis **(}**)

Printing the value of variable without declaring it

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

printf("%d", y));  // Printing the value of variable without declaring it

}
```

Output: Compile time error: syntax error , Printing the value of variable without declaring it

Printing the value of variable 'y' without declaring it

Logical errors

Sometimes, a programmer will write a statement where the syntax is perfectly correct and the translators translate the source code into object code correctly. However the wrong answer is

given when the program is run! This type of error is known as a '**logical error**'. They can be quite hard to spot unless a program is tested thoroughly.

Many different types of programming mistakes can cause logic errors.

For example,
assigning a value to the wrong variable may cause a unexpected program errors.
Multiplying two numbers instead of adding them together may also produce unwanted results.
Store the data at invalid array index.
Arithmetic error like divide by zero.

Example: Store the data at invalid array index.

```
#include<stdio.h>
void main()
{
   int a[5];
   a[10] = 100; // error,

   printf("%d", a[10]);          // error,
}
```

Output: error, array index out of bounds error.

This program will compile because syntax of program is correct but not run because size of array is 5 and we are trying to store the 100 at index 10.

Example: Arithmetic error like divide by zero.

```
#include<stdio.h>
void main()
{
   int a = 0;
   int b = 100/a;   // error, arithmetic error, divide by zero

}
```

Output: error, arithmetic error, divide by zero

This program will compile because syntax of program is correct but not run because value of b is not known at compile time but when we run it then value of b found infinite which is not be able to print so error will produce at run time. .

Decision Making Statement

In the C language decision making statement is executes if the given condition is true otherwise conditional block will never execute.

C language assumes **non-zero** and **non-null** values as **true**, and **zero** or **null** is assumed as **false** value.

There are following types of decision making statements in C programming language.

if statement

Syntax of an 'if' statement −

```
if(boolean_expression)
{
        /* statement will execute if the boolean expression is true */
}
```

In a 'if' statement if the Boolean expression is **true**, then the block of statement of the 'if' statement will be executed. If the Boolean expression is **false**, then the block of statement of the 'if' statement will not executed and control sent to the next line of if block.

Example: Write a program to demonstrate the 'if' statement.

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

        int x=10;
        int y=25;

        if(x<y)          // condition x<y is true therefore if block will execute
        {       printf("Sum of x+y = %i", (x+y));

        }
        printf("\n after if statement");

  }
```

Output: Sum of x+y = 35
after if statement

In the above program if reverse the ‘if’ condition of ‘if’ statement then the condition will be false and ‘if’ block will never execute.

```
include<stdio.h>
include<conio.h>
void main()
{
        int x=10;
        int y=25;

        if(x>y)          // condition x>y is false therefore if block will not execute
        {       printf("Sum of x+y = %i", (x+y));

        }
        printf("\n after if statement");

}
```

Output: after if statement

if-else statement

Syntax of an 'if-else' statement −

```
if(boolean_expression)
 {
    /* statement will execute if the boolean expression is true */
 }
else
{
}
```

In a ‘if-else’ statement if the Boolean expression is **true**, then the block of statements of the 'if' statement will be executed. If the Boolean expression is **false**, then the block of code of the 'else' statement will executed.

Example: Write a program to take two numbers from user and find the grater between them using ‘if-else’ statement.

```
include<stdio.h>
include<conio.h>
void main()
{
        int x;
        int y;

          printf("enter  1st number”);
          scanf(“%d”,&x);                    // 10
          printf("enter  2nd number”);
          scanf(“%d”,&y);                    // 30


        if(x>y)          // check condition x>y
        {       printf("grater value=  %d", x);

        }
        else
        {       printf("\n grater value=  %d", y);
        }

}
```

Output: enter 1st number
10 // user enter 10 in variable x
enter 2nd number
30 // user enter 30 in variable y
grater value= 30

Nested -if statement

In a C language we can use **if** statement inside another **if** statement(s).

Syntax of an 'nested -if' statement −

```
if(condition 1)
 {
        /* statement will execute if the condition 1 is true */

          If (condition 2)
          {         /* statement will execute if the condition 2 is true */
          }
```

```
}
```

Example: Write a program to take two numbers from user if both number are between 1 to 9 then print "Good" using nested-if statement.

```
include<stdio.h>
include<conio.h>
void main()
{
        int x;
        int y;

          printf("enter 1st number");
          scanf("%d",&x);                    // 5
          printf("enter 2nd number");
          scanf("%d",&y);                    // 7


        if(x<10)        // check condition x<10
        {

                if(y<10)                // nested-if, check condition y<10
                {       printf("Good");

        }

}
```

Output:
```
enter 1st number
5                       // user enter 5 in variable
enter 2nd number
7                       // user enter 7 in variable
 Good
```

Nested if-else statements

In a C language we can use **if** or **if-else** statement inside another **if** or **if-else** statement(s).

Syntax of an 'if-else' statement −

```
if(condition 1)
{
        /* statement will execute if the condition 1 is true */

          If (condition 2)
          {         /* statement will execute if the condition 2 is true */
          }
}
else
{
          If (condition 3)
          {         /* statement will execute if the condition 3 is true */
          }

}
```

Example: Write a program to take three numbers from user and find the grater among them using 'nested if-else' statement.

```
include<stdio.h>
include<conio.h>
void main()
{
   int n1, n2, n3;
          printf("Enter three numbers: ");
          scanf("%d %d %d", &n1, &n2, &n3);

          if (n1>=n2)
          {
                    if(n1>=n3)
                    {         printf(" the largest number= %d", n1);
                    }
                    else
                    {         printf (" the largest number= %d", n3);
```

```
                }
        }
        else
        {
                if(n2>=n3)
                {       printf(" the largest number= %d", n2);
                }
                else
                {       printf(" the largest number=%d",n3);
                }
        }
}
```

Output:

Enter three numbers:

20

10

30

the largest number= 30

Example: Write a C program to input basic salary of an employee and calculate gross salary according to given conditions.
Basic Salary <= 10000 : HRA = 20%, DA = 80%
Basic Salary is between 10001 to 20000 : HRA = 25%, DA = 90%
Basic Salary >= 20001 : HRA = 30%, DA = 95%

```
include<stdio.h>
include<conio.h>
void main()
{
   float basic, gross, da, hra;
```

```
                    /* Input basic salary of employee */
    printf("Enter basic salary of an employee: ");
    scanf("%f", &basic);


            /* Calculate D.A and H.R.A according to specified conditions */
    if(basic <= 10000)
    {
        da  = basic * 0.8;
        hra = basic * 0.2;
    }
    else
    {
                            if(basic <= 20000)
                            {
                            da  = basic * 0.9;
                            hra = basic * 0.25;
                            }
                            else
                            {
                            da  = basic * 0.95;
                            hra = basic * 0.3;
                            }
          }

    /* Calculate gross salary */
    gross = basic + hra + da;

    printf("GROSS SALARY OF EMPLOYEE = %.2f", gross);

    return 0;
}
```

Output:

Enter basic salary of an employee:22000

GROSS SALARY OF EMPLOYEE = 49500.00

Example: Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer, calculate percentage and grade according to given conditions:
If percentage >= 90% : Grade A
If percentage >= 80% : Grade B
If percentage >= 70% : Grade C

If percentage >= 60% : Grade D
If percentage >= 40% : Grade E
If percentage < 40% : Grade F

Logic to calculate percentage and grade

In primary mathematics classes you have learned about percentage. Just to give a quick recap, below is the formula to calculate percentage.

$$Percentage = \frac{part}{whole} \times 100$$

Step by step descriptive logic to find percentage and grade.

1. Input marks of five subjects in some variable say phy, chem, bio, math and comp.
2. Calculate percentage using formula per = (phy + chem + bio + math + comp) / 5.0;. Carefully notice I have divided sum with 5.0, instead of 5 to avoid integer division.
3. On the basis of per find grade of the student.
4. Check if(per >= 90) then, print "Grade A".
5. If per is not more than 90, then check remaining conditions mentioned and print grade.

```
include<stdio.h>
include<conio.h>
void main()
{
    int phy, chem, bio, math, comp;
    float per;

                    /* Input marks of five subjects from user */
    printf("Enter five subjects marks: ");
    scanf("%d%d%d%d%d", &phy, &chem, &bio, &math, &comp);


                    /* Calculate percentage */
    per = (phy + chem + bio + math + comp) / 5.0;

    printf("Percentage = %.2f\n", per);
```

```
                /* Find grade according to the percentage */
   if(per >= 90)
   {
      printf("Grade A");
   }
   else
   {                if(per >= 80)
                    {          printf("Grade B");
                    }
                    else
                    {          if(per >= 70)
                         {     printf("Grade C");
                               }
                               else
                               {          if(per >= 60)
                                          {          printf("Grade D");
                                          }
                                          else
                                          {          if(per >= 40)
                                                     {          printf("Grade E");
                                                     }
                                                     else
                                                     {          printf("Grade F");
                                                     }
                                          }

                               }

                    }

          }

}
```

Switch statement

“switch” is a case control structure.

It contains case and default values.

“switch” statement takes a value as a expression for equality testing against a list of values/case.

syntax for a **switch** statement:-

```
switch(expression)
{
  case constant-expression  :
                                    statement(s);
                                    break;              /* optional */

  case constant-expression  :
                                    statement(s);
                                    break;              /* optional */

  /* you can have any number of case statements */
  default :                                             /* Optional */
                                     statement(s);
}
```

Rules of **switch** statement −

In a “switch” statement we can have any number of case.

for equality testing against a list of values/case, **constant-expression** for a case must be the same data type as the variable in the switch expression.

When “switch” expression value is equal to a case, the statements following that case will execute until a **break** statement is reached/executes.

When a **break** statement is reached/executes, the switch statement terminates, and the control jumps to the outside of switch.

a **break is a optional**. If there is no **break in the switch case**, the flow of control will jump to the next case of switch until a break is reached.

A **switch** statement also have a **default** case, which appear at the end of the switch. when none of the cases is true then default case will execute. No **break** is needed in the default case. “default” case is also an optional.

Example: Write a program to take one numbers from user between 1 to 3. Write entered number in words if number between 1 to 3 otherwise print wrong number using switch.

```
include<stdio.h>
include<conio.h>
void main()
{       int n;
        printf(" enter number between 1 to 3");
        scanf("%d", &n);

        switch(n)
        {       case 1 :
                        printf("\n one" );
                        break;
                case 2 :
                        printf("\n two" );
                        break;
                case 3 :
                        printf("\n three" );
                        break;

                default :
                        printf("\n wrong number" );

          }

}
```

Output:

```
enter number between 1 to 3
2                                   // user entered 2
two
```

Note: In the above program, for example user enters a number 6 then output will be as follows:

Output:

```
enter number between 1 to 3
6                                   // user entered 2
wrong number
```

Example: Write a program using switch statement but without break.

```
#include <stdio.h>
void  main ()
{
char c = 'B';

        switch(c)
         {
            case 'A' :
                        printf("Excellent!\n" );
                        break;
            case 'B' :
            case 'C' :
                        printf("Well done\n" );
            case 'D' :
                        printf("You passed\n" );
                        break;
            case 'F' :
                        printf("Better try again\n" );
                        break;
            default :
                        printf("Invalid Character  \n" );
        }
}
```

Output

```
Well done
You passed
```

Description: Because if there is no **break in the switch case**, the flow of control will jump to the next case of switch until a break is reached.

Note: In the above program, for example value of char variable c = ‘ A’ then the output will be as follows:

Output

Excellent!

Nested switch statements

In C programming language we can define switch statement within another switch.

Syntax for a **nested switch** statement: −

```
switch(ch1)
{   case 'A':
                        printf("case A of outer switch" );

     switch(ch2)
    {
       case 'A':
            printf("case  A of inner switch" );
            break;
       case 'B': /* case code */
     }

                        break;

  case 'B': /* case code */
}
```

Example: C program to read weekday number and print weekday name using switch

This program will read **weekday number (0-6) and print weekday name (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday)** according to given weekday number

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

void main()
{
   int wDay;

   printf("Enter weekday number (0-6): ");
   scanf("%d",&wDay);

   switch(wDay)
   {
      case 0:
         printf("Sunday");
         break;
      case 1:
         printf("Monday");
         break;
      case 2:
         printf("Tuesday");
         break;
      case 3:
         printf("Wednesday");
         break;
      case 4:
         printf("Thursday");
         break;
      case 5:
         printf("Friday");
         break;
      case 6:
         printf("Saturday");
         break;
      default:
         printf("Invalid weekday number.");
   }
   printf("\n");
   return 0;
}
```

```
Output:
Enter weekday number (0-6): 3       // 3 enter by user
```

Wednesday

The ? : Operator

This operator also known as **conditional operator .** It is similar to if-else statement**.**

Syntax of ? operator

Expression1 ? Expression 2 : Expression 3;

There are three expressions in ? operator, Expression1, Expression2, and Expression3 .

The value of a ? expression is determined like this −

- If Exp1 is true, then Exp2 is evaluated.
- If Exp1 is false, then Exp3 is evaluated.

Example: write a program to find the grater between two number using ? operator.

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

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

        printf("\ngrater value = %d", c);

}
```

Output: grater value = 20

Example: write a program to find the grater among three number using ? operator.

```
include<stdio.h>
include<conio.h>
void main()
{
  int a, b, c, big ;
   printf("Enter three numbers : \n ") ;

 scanf("%d %d %d", &a, &b, &c) ;

 big = a > b ? (a > c ? a : c) : (b > c ? b : c) ;
```

```
printf("\nThe biggest number is : %d", big) ;
}
```

Output: Enter three numbers :

```
10                    // 10 enter by user
33                    // 33 enter by user
20                    // 20 enter by user
```

Loops

In a programming often a situation may comes where we want to execute the set of code again and again.

Loop has a ability to repeat set of statement until a condition to be satisfied or a particular number of times. There are following types of loop in C language:

In a C programming language we can also define a loop within another loop.

- while loop
- do...while loop
- for loop
- nested loops

while loop

"while loop" repeats a set of statements if condition is true. If condition is false the flow of control do not enter inside of while loop. It first tests the condition then execute the body of loop.

Syntax of a **while** loop −

```
while(condition)
{
  Statement1;
  statement2;
    ----
    ----

Increment/Decrement;
}
```

Example: Write a program to print 1 to 10 using while loop.

```
#include <stdio.h>
int main ()
{           int a = 1;

                    while( a <= 10 )          //  loop executes 10 times
                    {           printf(" %d \n", a);

                                a++;
                    }
return 0;

}
```

Output: 1
2
3
4
5
6
7
8
9
10

Description: In the above program, line “while(a <= 10)” will check if condition is true or false. In the above case when a=1, condition is true and flow of control move inside of while and print 1 after that increment occurs.

Now a=2, again condition is true and flow of control move inside of while and print 2 after that increment occurs.

Now a=3, again condition is true and flow of control move inside of while and print 3 after that increment occurs.

This process is continue until condition is false(in this case condition false occur when a=11).

Example: Write a program to print 10 to 1 using while loop.

```
#include <stdio.h>
int main ()
{           int a = 10;

                    while( a >0 )   //  loop executes 10 times
                    {           printf(" %d \n", a);

                                a--;
                    }
return 0;

}
```

Output: 10
9
8
7
6
5
4
3
2
1

Example: WAP to take input from user and find a factorial using a while loop in C?

```
#include<stdio.h>
int  main()
{
        int num, i, fact=1;
        printf("Enter the number\n");
        scanf("%d",&num);
                while(num>=1)
                {       fact=fact*i;
                        i--;
                }
```

```
printf("The factorial of given number= %d ",fact);
}
```

Output: Enter the number

5 // 5 entered by user

The factorial of given number= 120

Example: WAP to take input from user and reverse the number using a while loop in C?

```c
#include <stdio.h>
int main()
{
    int n, reversedNum = 0, remainder;

    printf("Enter an integer: ");
    scanf("%d", &n);

    while(n != 0)
    {
        remainder = n%10;
        reversedNum = reversedNum*10 + remainder;
        n = n/10;
    }

    printf("Reversed Number = %d", reversedNum);

}
```

Output: Enter an integer: 2345
Reversed Number = 5432

Example: C program for Fibonacci series up to given length.

```c
#include<stdio.h>
#include<conio.h>
main()
{
   int a=0,b=1,c,i=3,len;
   printf("enter length of the  fibonacci series:");
   scanf("%d",&len);
   printf("%d\t%d",f1,f2); // It prints the starting two values
```

```
   while(i<=len)          // checks the condition
   {
      c=a+b;                                  // performs add operation on previous two  values
      printf("%d",c);                          // It prints from third value to given length
      a=b;
      b=c;
      i=i+1;                // incrementing the i value by 1
   }
   getch();
}
```

Output: enter length of the fibonacci series:10

0 1 1 2 3 5 8 13 21 34

do-while loop

like a while loop, " do while loop" repeats a set of statements if condition is true. If condition is false the flow of control do not enter inside the do-while loop.

"do-while" first execute the body of loop than tests the condition.

Means even if the condition if false do-while executes at least ones.

The syntax of a **do...while** loop in C programming language is −

```
do {
  Statement1;
   statement2;
      ----
      ----

Increment/Decrement;
} while( condition );
```

Example: Write a program to print 1 to 10 using while loop.

```
#include <stdio.h>
int main ()
{           int a = 1;
```

```
                do
                {          printf(" %d \n", a);

                          a++;
                } while( a <=10 );       //  loop executes 10 times
return 0;

}
```

Output: 1
2
3
4
5
6
7
8
9
10

Even if the condition if false do-while executes at least ones.

```
#include <stdio.h>
int main ()
{           int a = 100;

                do
                {          printf(" %d \n", a);

                          a++;
                } while( a <=10 );       //  condition is false even do-while executes ones
return 0;

}
```

Output: 100

Description: in the above program do-while executes ones even condition of is false. Because do-while first executes the statement then check the condition.

for loop

“for-loop” repeat the set of statement until a condition to be satisfied or a particular number of times.

Syntax of for-loop:

```
for(initialization; condition; iteration )
{       statement 1;

        statement 2;

        ----------
```

“for-loop” has a initialization , condition and iteration (increment/decrement) part separated by semicolon.

Initialization : This phase allows you to initialize loop control variables.

Condition : If condition of for-loop is true, then the body of the loop will execute. And the body of the loop does not execute if the condition is false, and the flow of control jumps to the next statement just after the 'for' loop

Iteration phase: This statement allows you to update(increment/decrement any loop control variables..

Example: Write a program to print 1 to 10 using for-loop.

```
#include <stdio.h>
int main ()
{       int a;

        for( a = 1; a <= 10; a++ )
        {
            printf(" %d  \n", a);
        }
```

return 0;

}

Output: 1
2
3
4
5
6
7
8
9
10

In a for loop either initialization or iteration , or both may be absent but condition must be present in a for loop.

```
#include <stdio.h>
int main ()
{           int a=1;

                    for(   ; a <= 10;   )
                    {
                        printf(" %d  \n", a);
                        a++;
                    }
return 0;

}
```

Output: 1
2
3
4
5
6
7

8
9
10

Example: Write a program to take a number from user and find its factorial.

```
#include <stdio.h>
 int main()
{
  int i, n, fact = 1;
  printf("Enter a number to calculate its factorial\n");
  scanf("%d", &n);

        for (i = 1; i <= n; i++)
        {       fact = fact * i;
        }
  printf("Factorial = %d\n",  fact);

  return 0;
}
```

Output:

Enter a number to calculate its factorial
5
Factorial = 120

Example: Write a program to take a number from user and check whether an number (entered by the user) is a prime number or not.

Prime Number : prime number is a number which satisfies the following conditions.

- It should be whole number
- It should be greated than 1
- It should have only 2 factors. They are, 1 and the number itself.

Example for prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, 23,29 etc. because this numbers is only divided by 1 and the number itself.

Number 4, 6, 8, 9, 10, 12, 14, 15, 16 etc are not prime numbers. Because, the number 4 can be divided by 2. As per the rule of prime number, should be divide 2 numbers only. They are 1 and the number itself. But, number 4 is also divided by 2. Similarly , all remaining numbers 6, 8, 9, 10, 12, 14…….. also divided by a number other than 1 and the number itself. Therefore these are not a prime numbers.

Number 1 is neither a prime nor a composite number.

```
#include <stdio.h>
int main()
{
   int n, i, flag = 0;
        printf("Enter a number to check Whether a Number is Prime or Not: ");
        scanf("%d", &n);

        for(i = 2; i <= n/2; ++i)
        {
                if(n%i == 0)    // condition for nonprime number
                {       flag = 1;
                        break;
                }
        }

        if (n == 1)
        {       printf("1 is neither a prime nor a composite number.");
        }
        else
        {       if (flag == 0)
                {       printf(" Entered number is a prime number.");
                }
                else
                {       printf("Entered number is not a prime number.");
                }
```

```
        }

   return 0;
}
```

Output:

Enter a number to check Whether a Number is Prime or Not:

7 // enter by user

Entered number is a prime number.

Example: C program for Fibonacci series up to given length.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a=0,b=1,c,i=3,len,i;
   printf("enter length of the  fibonacci series:");
   scanf("%d",&len);
   printf("%d\t%d",f1,f2);                    // It prints the starting two values
        for( i=1; i<len; i++)                  // checks the condition
        {
                c=a+b;                         // performs add operation on previous two  values
                 printf("%d",c);               // It prints from third value to given length
                 a=b;
                b=c;
        }
   getch();
}
```

Output: enter length of the fibonacci series:10

0 1 1 2 3 5 8 13 21 34

Program for pattern printing

Program to print half pyramid using *

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

Source Code

```
#include<stdio.h>
int main() {
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (j=1; j<=i; ++j)
        { printf("* "); }
        printf("\n");
    }
    return 0;
}
```

Program to print half pyramid a using numbers

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Source Code

```
#include<stdio.h>
int main() {
    int i,j,rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (j=1; j<=i; ++j)
        { printf("%d ",j); }
        printf("\n");
    }
    return 0;
```

```
}
```

Program to print half pyramid using alphabets

```
A
B B
C C C
D D D D
E E E E E
```

Source Code

```
#include<stdio.h>
int main() {
    int i, j;
    char input, alphabet='A';
    printf("Enter the uppercase character you want to print in last row: ");
    scanf("%c", &input);
    for (i=1; i<=(input-'A'+1); ++i) {
        for (j=1; j<=i; ++j)
        { printf("%c", alphabet); }
        ++alphabet;
        printf("\n");
    }
    return 0;
}
```

Programs to print inverted half pyramid using * and numbers

Inverted half pyramid using *

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

Source Code

```c
#include<stdio.h>
int main() {
   int i, j, rows;
   printf("Enter number of rows: ");
   scanf("%d", &rows);
   for (i=rows; i>=1; --i) {
      for (j=1; j<=i; ++j)
      { printf("* "); }
      printf("\n");
   }
   return 0;
}
```

Question: Write a program to print design.

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

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

Question: Write a program to print design.

```
1
22
333
4444
```

55555

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

Question: Write a program to print design.

1
12
123
1234
12345

```
#include<conio.h>
#include<stdio.h>
void main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1; j<=i; j++)
        {
```

```
            printf("%d", j );
        }
    printf("\n");
    }
getch();
}
```

Question: Write a program to print design.

1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

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

Question: Write a program to print design.

*****

****

***

**

*

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

Question: Write a program to print design.

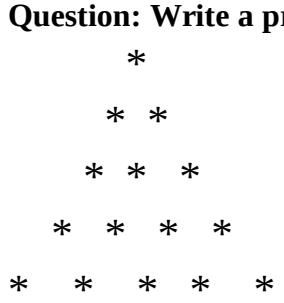
```
     *
    **
   ***
  ****
 *****
```

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

    printf("\n");
    }

getch();
}
```

Question: Write a program to print design.

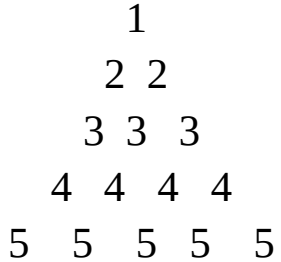


```
#include<conio.h>
#include<stdio.h>
void main()
{
    int i,j,k;
    for(i=1;i<=5;i++)
    {          for(k=i; k<5; k++)
               {
                 printf(" ");
                }
                for(j=1; j<=i; j++)
               {
                 printf("*  " );
               }


    printf("\n");
    }

getch();
}
```

Question: Write a program to print design.



```
#include<conio.h>
#include<stdio.h>
void main()
{
    int i,j,k;
    for(i=1;i<=5;i++)
    {          for(k=i; k<5; k++)
               {
                printf(" ");
               }
               for(j=1; j<=i; j++)
               {
                printf(" %d", i );
               }


    printf("\n");
    }

getch();
}
```

Question: Write a program to print design.

```
          1
        1  2
      1  2  3
   1   2   3   4
1    2    3   4    5
```

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

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

getch();
}
```

Question: Write a program to print design.

```
          1
        2  3
      4  5  6
   7   8   9   10
11  12  13  14   15
```

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

    printf("\n");
    }
getch();
}
```

Inverted half pyramid using numbers

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Source Code

```
#include<stdio.h>
int main() {
    int i ,j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=rows; i>=1; --i) {
        for (j=1; j<=i; ++j)
        { printf("%d ",j); }
        printf("\n");
    }
    return 0;
}
```

Programs to display pyramid and inverted pyramid using * and digits

Program to print full pyramid using *

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

Source Code

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i,k=0) {
        for (space=1; space<=rows-i; ++space)
```

```
        { printf("  "); }
        while (k!=2*i-1) {
           printf("* ");
           ++k;
        }
        printf("\n");
    }
    return 0;
}
```

Program to print pyramid using numbers

```
    1
  2 3 2
 3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

Source Code

```
#include<stdio.h>
int main() {
    int i, space, rows, k=0, count=0, count1=0;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=1; i<=rows; ++i) {
        for (space=1; space<=rows-i; ++space) {
          printf("  ");
          ++count;
        }
        while (k!=2*i-1) {
           if (count <= rows-1)
           { printf("%d ", i+k);
             ++count;
           }
           else {
             ++count1;
             printf("%d ", (i+k-2*count1));
           }
           ++k;
        }
        count1=count=k=0;
        printf("\n");
    }
    return 0;
}
```

Inverted full pyramid using *

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

Source Code

```
#include<stdio.h>
int main() {
    int rows, i, j, space;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=rows; i>=1; --i) {
        for (space=0; space<rows-i; ++space)
            printf("  ");
        for (j=i; j<=2*i-1; ++j)
            printf("* ");
        for (j=0; j<i-1; ++j)
            printf("* ");
        printf("\n");
    }
    return 0;
}
```

Print Pascal's triangle

```
      1
     1  1
    1  2  1
   1  3  3  1
  1 4  6  4  1
 1 5  10  10 5  1
```

Source Code

```
#include<stdio.h>
int main() {
    int rows, coef=1, space, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    for (i=0; i<rows; i++) {
```

```
7.        for (space=1; space <= rows-i; space++)
8.            printf("  ");
9.        for (j=0; j<=i; j++) {
10.           if (j==0 || i==0)
11.               coef = 1;
12.           else
13.               coef=coef*(i-j+1)/j;
14.           printf("%4d", coef);
15.       }
16.       printf("\n");
17.   }
18.   return 0;
19. }
```

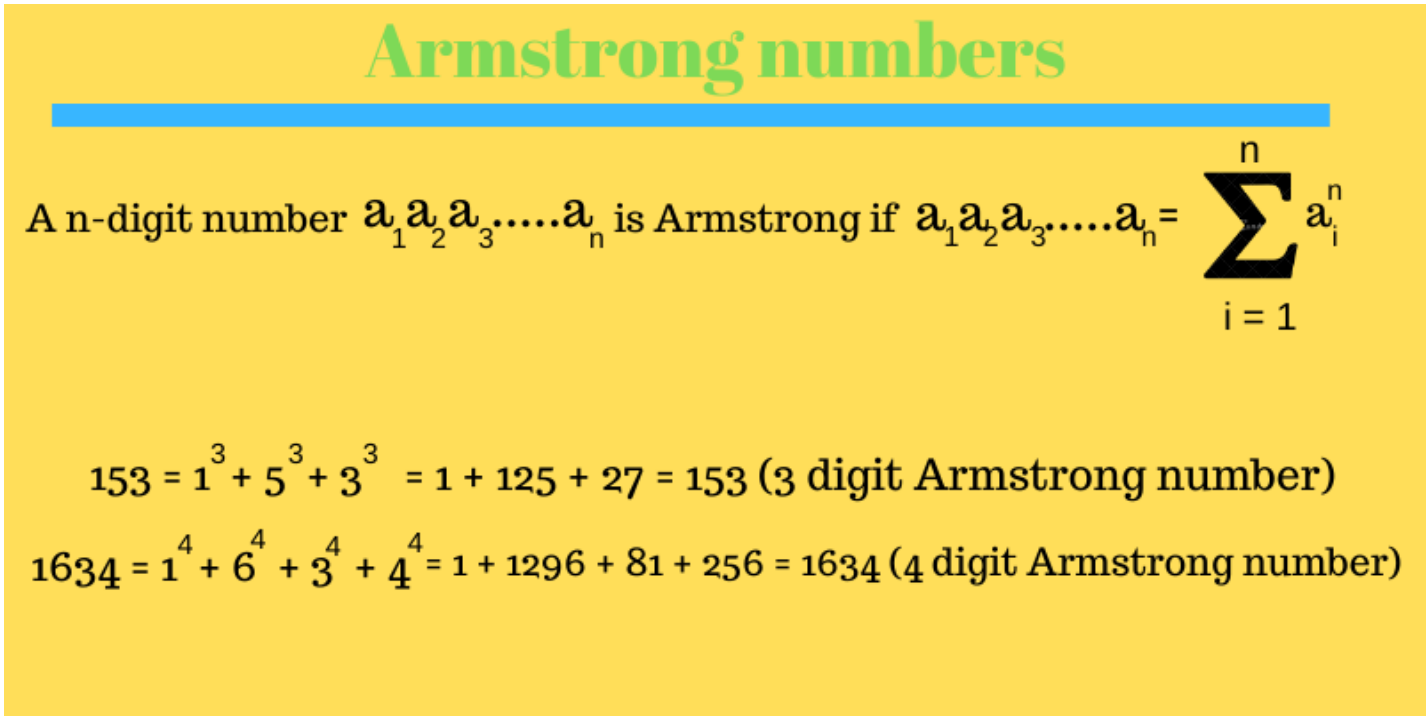
Print Floyd's Triangle.

```
1
2 3
4 5 6
7 8 9 10
```

Source Code

```
1.  #include<stdio.h>
2.  int main() {
3.      int rows, i, j, number= 1;
4.      printf("Enter number of rows: ");
5.      scanf("%d", &rows);
6.      for (i=1; i<=rows; i++) {
7.          for (j=1; j<=i; ++j)
8.          { printf("%d ", number);
9.            ++number;
10.         }
11.         printf("\n");
12.     }
13.     return 0;
14. }
```

Armstrong number in C



Armstrong number program in C

```
#include <stdio.h>
int main()
{
  int n, sum = 0, t, remainder, digits = 0;

  printf("Input an integer\n");
  scanf("%d", &n);

  t = n;
  // Count number of digits
  while (t != 0) {
    digits++;
    t = t/10;
  }

  t = n;

  while (t != 0) {
    remainder = t%10;
    sum = sum + remainder* remainder* remainder;
    t = t/10;
  }
```

```
  if (n == sum)
    printf("%d is an Armstrong number.\n", n);
  else
    printf("%d isn't an Armstrong number.\n", n);

  getch();

  return 0;
}
```

Leap year C program

```
 #include <stdio.h>

int main()
{
  int year;

  printf("Enter a year to check if it's a leap year\n");
  scanf("%d", &year);

  if (year%400 == 0) // Exactly divisible by 400, e.g., 1600, 2000
    printf("%d is a leap year.\n", year);
  else if (year%100 == 0) // Exactly divisible by 100 and not by 400, e.g., 1900, 2100
    printf("%d isn't a leap year.\n", year);
  else if (year%4 == 0) // Exactly divisible by 4 and neither by 100 nor 400, e.g., 2016, 2020
    printf("%d is a leap year.\n", year);
  else // Not divisible by 4 or 100 or 400, e.g., 2017, 2018, 2019
    printf("%d isn't a leap year.\n", year);

  getch();

  return 0;
}
```

Strong Number in C

Here you will get program for strong number in C.

What is Strong Number?

A number in which the sum of factorial of individual digits is equal to the number is called strong number.

For example, 145 is a strong number because 145=(!1)+(!4)+(!5)=1+24+120=145

The below C program will check whether a given number in strong number or not.

```
#include<stdio.h>

int main(){
	int n,t,sum,m,fact,i;

	printf("Enter a number:");
	scanf("%d",&n);

	m=n;

	while(m!=0){
		t=m%10;
			fact = 1;

      for(i=1; i<=t; i++)

		fact = fact * i;

			sum+=fact;
			m=m/10;
	}

	if(sum==n){
		printf("Strong Number");
	}
	else{
		printf("Not Strong Number");
	}

	return 0;
}
```

C Program to find Perfect Number

Perfect Number in C

If the sum of its positive divisors excluding the number itself is equal to that number then number is called perfect number.

For example, 6 is a perfect number in C because 6 is divisible by 1, 2, 3 and 6. So, the sum of these values is 1+2+3 = 6 (Remember, we have to exclude the number itself. That's why we haven't added 6 here). Some of the perfect numbers are 6, 28, 496, 8128 and 33550336 etc

/* C Program to find Perfect Number using For Loop */

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

```
int main()
{
 int i, Number, Sum = 0 ;

 printf("\n Please Enter any number \n") ;
 scanf("%d", &Number) ;

 for(i = 1 ; i < Number ; i++)
  {
   if(Number % i == 0)
     Sum = Sum + i ;
  }

 if (Sum == Number)
    printf("\n %d is a Perfect Number", Number) ;
 else
    printf("\n%d is not the Perfect Number", Number) ;
getch();
return 0 ;
}
```

Jump Statements

Jump statements are used to interrupt the normal flow of program.

Types of Jump Statements

- Break
- Continue
- GoTo

The break statement is used inside loop or switch statement. When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example of break statement

```
void main()
{   int a=1;
```

```
                while(a<=10)
                 {          if(a==5){
                              break;
                            }
                           printf("\n %d.",a);
                       a++;
                   }
    }
```

Output : 1
2
3
4

continue statement

The continue statement is also used inside loop. When ever we want to give the control back to the beginning of the loop we can do so by using the “continue” .

Example of continue statement

```
void main()                                                                    {
int a=1;
                while(a<=10)
        {       if(a==5)
        {     continue;
                          }
              printf(" %d.",a);
                 a++;
      }                                                                        }
```

Output 1 2 3 4 6 7 8 9 10

In this program when the value of variable ‘a’ is 5 then ‘continue’ statement give the control back to the beginning o f the loop so 5 will not print.

goto statement

The goto statement is a jump statement which jumps from one point to another point within a function.

Syntax of goto statement

```
goto label;
    ----------
    ----------
label:
    ----------
    ----------
```

In the above syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code after it.

Example of goto statement

```
#include<stdio.h>

void main()
{
        printf("\nStatement 1.");
        printf("\nStatement 2.");
        printf("\nStatement 3.");

        goto last;

        printf("\nStatement 4.");
        printf("\nStatement 5.");

        last:

        printf("\nEnd of Program.");
}
```

Output :

```
Statement 1.
Statement 2.
Statement 3.
End of Program.
```