C Programming

C/C++

Lotus It Hub

“We care for your career”

ISO 9001:2015 Certified

2nd Floor, Krishna Tower, Near Main Karve Nagar Bus Stop, Pune. Contact No: 9730258547/ 8483966654Visit Us: www.lotusithub.com facebook/lotusithubpune

C and C++

C language

Basic c language:

C is middle level language. And this is first programming level language of software.

BCPL= “Basic C Programming Language”=1969;

B= “Basic” = 1970;

C= 1972= developed by Dennis Ritches;

Computer:

It is electronic device which perform input and output task.

Type: 1) hardware :

1: CPU

2: keyboard

3: mouse

2) software :

1: System software

2:Application software

3:Drive software

1. **System s/w=** its compulsion part of system. Windows 7 ,8,xp
2. **Application s/w=** example antivirus, turbo c, ms office
3. **Drive s/w=** Bluetooth , keyboard, PD.

Program : How to display “Hellow” word in C Language.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

printf(“Hellow”);

getch();

}
```

\\output: Hellow

Parameter information:

1) **#** : It is pre processor directive. Which is used is a compiler of code.

2) **include** : It is folder which contain input and output data.

3) **stdio.h** : Standard input output

Functions used in it : **printf( )** and **scanf( )**

It is a header file ,which contain input and output data.

4) **.h extension** : Header file

5) **conio.h** : console input and output

Functions used in it : **getch( )** and **clrscr( )**

6) **void** : It is key word which does not return any where.

7) **main( )** : It is function and it is set of instruction.

main is starting of program , it does not terminate with semi colon (**;**)

8) **printf( )** : It is used for displaying output

9) **scanf( )** **:** It is used for taking input from user

10) **clrscr( )** **:** Clear screen previous data

11) **getch( )** : It is hold the consolation in program

12) **\n** : For new line

13) **\t** : For space

Data Types

The following table provides the details of standard integer types with their storage sizes and value ranges –

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
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
Float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
Double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Program: write program display first line Hello then new line Lotus It Hub.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

printf(“Hello \n Lotus\tIt \t Hub”);

getch();

}
```

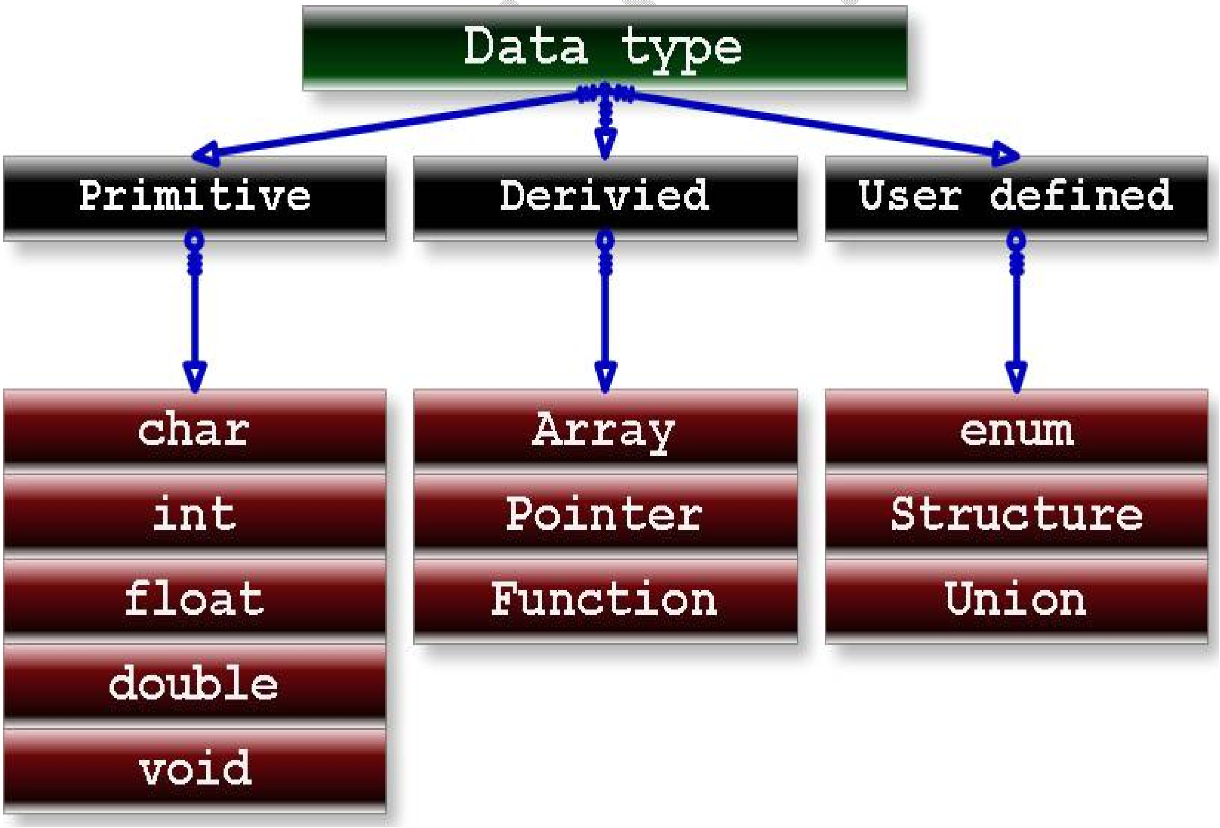
```
 \\output :        Hello

                   Lotus It Hub
```

Data type:

Data type refer to an extension system used for declaring variable or function of different types.

Data types its memory allocation which is store some data

Data type	Memory	Range
Integer (int)	2 byte	-32768 to 32767
Float (float)	4 byte	1.2E-38 to 3.4E+38
Character (char)	1 byte	-128 to 127



Variable:

Variable is memory allocation which is store some data.

Ex. a=100

a is store 100 value.

Program: write a program to display integer , float, and charcter value.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=100;
float b= 23.45;
char c= ‘a’;
printf(“%d %f %c”,a,b,c);
getch();
}                                        \\output: 100 23.45 a
```

- In int and float is variable is not used any char sign. Its value a syntax error.

    Ex.   int +=100;

    printf(“%d”,+);

program: write program size of integer , float and charcter.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("%u\n",sizeof(12));
printf("%u\n",sizeof(2.3f));
printf("%u\n",sizeof('a'));

getch();
}
```

\\output: 2

```
4

1
```

How to Declare Variables

```
int a=100;

int a1=122;

int _=122;
```

Not Correct variable

```
int 12=12;

int !=100;

int @=23;

int +=123;

int &=345;
```

Swapping:

Program : write a program in swapping by using third variable.

```
#include<stdio.h>

#include<conio.h>

void  main()

{

clrscr();

int a=100,b=200,c;

  c=a;//100

  a=b;//200

   b=c;//100
printf("%d %d"a,b);

getch();

}
```

Output: a=200,b=100;

Program: write a program using two variable only by swapping.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100,b=200;

a=a+b;

b=a-b;

a=a-b;

printf("%d %d",a, b);

getch();

}
```

Output : a=200,b=100

Reverse number:

Program: write a program of reverse number of 3 digits/ 4 digits.

```
int a=123,b,c,d,e,f;

  b=a%10 ;//3

  c=a/10 ;//12

  d= c% 10 ;// 2

  e=c/10;//1

 f=b* 100+d*10+ e;
```

Program :

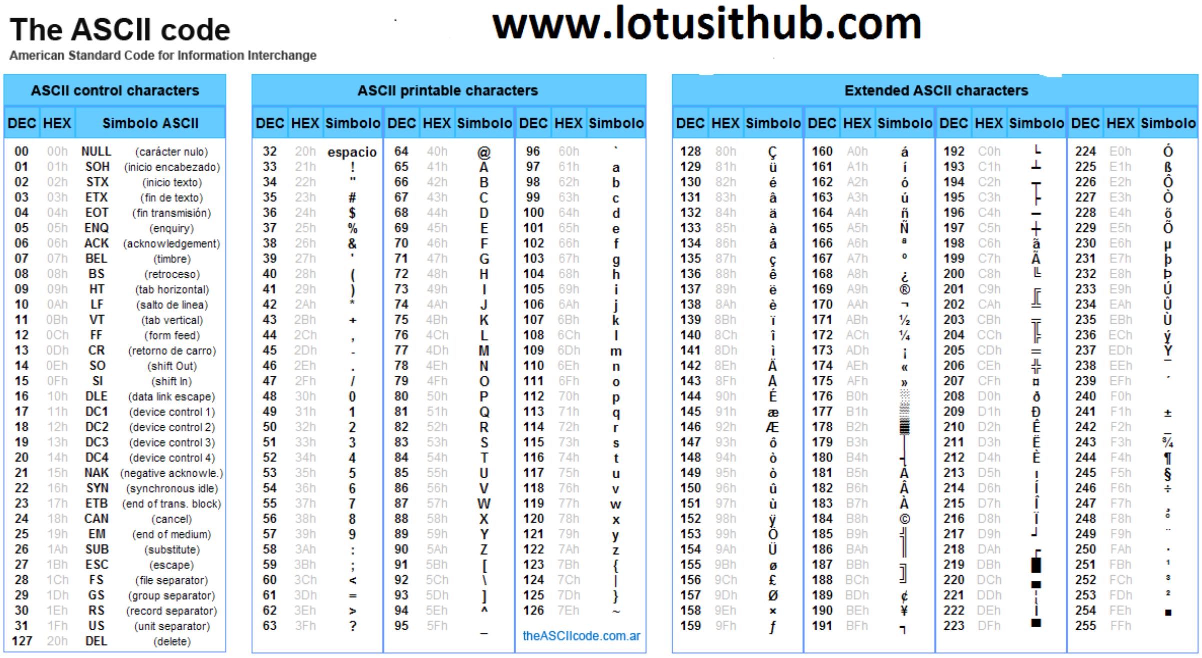
```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a,b,c,d,e,f;

 scanf("%d"&a);

b= a%10;

c=a/10;

d=c%10;

e=c/10;

f=b*100+d*10+e;

printf("%d",f);

getch();

}
```

Program: write a program of find total salary of employee where the basic salary of employee bs 20000 , hre =5%, tra= 6%, pf=7%, acc=8%.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int bs=20000;

float  total,hre, tra, pf, acc;

hre=bs*5/100;//1000

tra=bs*6/100;//1200

acc=bs*8/100;//1600

pf=bs*7/100;//1400

total=bs+hre+tra+acc-pf;

printf("%f", total);

getch();

}
```

Output :-22400

ASCII Code



The ASCII code

American Standard Code for Information Interchange

www.lotusithub.com

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

OPERATORS

An operator is symbol that tell the compiler to perform specific mathematical or logical manipulation c language is rich in built-in operators and provide the following types of operators:

1. Arithmetic
2. Logical
3. Increment/decrement
4. Bitwise
5. Conditional
6. Procedure association
7. Assignment

1. Arithmetic operators:

This operators used to all mathematical operation executed.

Program: write a program all arithmetic operation.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a,b,c,d,e,f,g;

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

c=a+b;

d=a-b;

e=a*b;

f=a/b;

g=a%b;

printf("% d\n %d\n %d\n %d\n %d",c,d,e,f,g);

getch();

}
```

```
\\output: a=15

          b=12

          27

          3

           180

           1

            3
```

2.logical operators:

AND (&&), OR(||), NOT(!)

Program: write a program find the greater number in 3 no.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
scanf("%d %d %d",&a ,&b ,&c);
if(a>b&&a>c)
{
printf("a is greater than b and c");
}
else if(b>a&&b>c)
{
printf("b is greater than a and c");
}
else if(c>a&&c>b)
{
printf("c is greater than a and c");
}
getch();
}
```

```
\\output: a=10, b=20, c=30;
             C is greater than a and b
```

3. Increment/decrement:

They have two types : 1. Pre increment/decrement

2. Post increment/ decrement

1.pre increment(++a) & pre decrement(--a)

2. post increment(a++)& post decrement(a--)

Example: b=10;

```
            a=b++;
  output:  a=10, b=11;
```

Problem: write a program post increment. If a=10.

```
#include<stdio.h>
Include<conio.h>
void main()
{
 clrscr();
  int a,b;
  a=10;
   b=a++;
    printf("%d  %d",a,b);
 getch();
     }
```

\\output : a=11, b=10. (a++)

```
         A=9,b=10. (a--)
           a=11, b=11. (++a)
            a=9, b=9 (--a)
```

program: write a program for pre and post increment /decrement.

```
 #include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a;
a= 10;
printf("%d %d %d %d %d %d ",++a,a++,a--,--a,++a,a++);
getch();
}
```

\\output: 12 10 11 11 12 10

- This program execute from right to left.

4. **Bitwise :**
   Bitwise operator works on a bits and its perform bit-by-bit operation.
   Its work in a AND, OR, XOR, NOT, left shift(<<), right shift(>>) operator.

   1. Left shift operator : it's a multiplication operator.

Problem: write a program for left shifting if a=10.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a;
a=10<<0;
printf("%d",a);
getch();
}
```

```
\\output : a:   0     1     2     3    4
          Ans: 10    20   40    80   160
```

2.right shifting(>>): it's a division operator .

Problem: write a program for right shifting if a=6.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a;
a=6>>2;
printf("%d",a);
getch();
}
```

```
\\output: 0    1       2    3
           6    3      1    0
```

5.conditional operator:

It is operator to used in a relation operator. Such as <,>, <=, >= etc.

Program: write a program find greater number in two variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a,b;
scanf("%d %d ",&a,&b);
if(a<b)
{
printf("a less than b");
}
else
{
printf("a is greater than b");
}
getch();
}
```

```
\\output:
a=10,b=20=> a less than b,
a=40, b=10=> a greater than b;
```

Program: write a program for find vowel and consonant .

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char a;
scanf("%d" &a);
if(a=='a'||a=='e'||a=='i'||a=='o'||a=='u')
{
printf("a is vowel");
}
else
{
printf("a is consonant");
}
getch();
}
```

\\output : a=a => a is vowel
a=b => a is consonant

program : write a program for capital and small alphabet.

In a ASCII code

Capital value A=65, Z=90;

Small value a=97, z=122;

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
scanf("%d",&a);
if(65<=a&&a<=90)
{
printf("a is capital");
}
else if (97<=a&& a<=122)
{
printf("a is small");
}
getch();
}
```

\\output: 67 => a is capital.
121 => a is small.

5. **Procedure association :** this operator is used to arithmetic operation .
   Priority of operation in computer
   *** / % and + -**
   It support left to right approach and priority of * , / and **%** are same

   Example:
   1. 10*20/3%1
      200/3%1
      66.6%1
      0

   2. 100+5-2*6/3%1

      100+5-12/3%1

      100+5-4%1

      100+5-0

      105

   3. 10-5*10-5

      10-50-5

      10-55

      -45

7. assignment operator:

Example : a=a+b; a+=b

Similarly we can used to all arithmetic operators such as

-=, *=, /=, <<=, >>=, %=,

And also work at logical gate such as

&=,^=, |= etc.

- **Ternary code:**

  It is used to reduce the code .

  Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a,b,c;
a=100, b=500;
c=a>b?a:b;
printf(“%d”, c);
getch();
}
```

Write a program find Greater no in three variables a=1000,b=500,c=200;
Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a,b,c,d;
a=1000, b=500,c=200;
d=(a>b?(a?c:a:c):(b>c?b:c));
printf(“%d”, d);
getch();
}
```

SWITCH CASE

A switch statement allows a variable to be tested for equality against a list of values. Each equality against a list of values . each value is called “case”. And the variable being switched on checked for each switch case.

Switch case is used to top to bottom jump.

- **Break** : this function is used to break the program is execution time.

Syntax:

```
switch(expression)

{

case constant-expression :

         statement ;

          break ;

           .

           .

default :

statement ;

}
```

Program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=1;
switch (a)
{
case 1:
{
printf(“Hellow\n”);
}
case 2:
{
printf("lotus it hub \n");
}
case 3:
{
printf("karve nagar\n");
}
default :
{
printf("pune")
}
}
getch();
}
   \\output : a=1 => Hellow
                     Lotus it hub
                     Karve nagar
```

Write a Program find addition or subtract, multiplication or division

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=1,b=200,c=100,d;
switch (a)
{
case 1:
{
d=b+c;
printf("Addition %d\n",d);
break;
}
case 2:
{
d=b-c;
printf("Subtract %d \n",d);
break;
}
case 3:
{
d=b*c;
printf(“Multiplication %d \n”,d);
break;

}
case 4:
{
d=b/c;
printf(“Division %d \n”,d);
break;

}

default :
{
printf(“Lotus it Hub”);
}
}
getch();
}
```

\\output : a=1 => Addition 300

a=2 => lotus it hub
karve nagar.

% If we give only “hellow” output then we used break statement as case 1 such as

```
switch (a)
{
case 1:
{
printf(“hellow\n”);
break ;
}
```

% In program the default statement in starting of program and you gives the value of “a” is greater than case number then its print default statement as well as all case statement .

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=10;
switch (a)
{
default :
{
printf("pune");
}
case 1 :
{
printf("Hellow\n");
}
case 2:
{
printf("lotus it hub");
}
}
getch();
}
```

```
\\output : a=10=> pune Hellow
                  lotus  it hub.
```

Note :duplicate case not allowed and only Consonant Value allow

Loop

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:

Type of loop

1.for loop

2.while loop

3.do while

1. For Loop

Syntax of For loop

```
For(initiation ; test condition ; increment/decrement )

{

   body

}
```

With code

```
For (i=1; i<=5; i++)

{
    printf  ("Hello");
}
```

Output=>

```
Hello Hello  Hello  Hello  Hello
```

First Step of loop initiation

=>test condition

=>body

=>increment or decrement

Once required initiation then test condition and increment /decrement

Program : write a program “hello” is printed 3 times

```
#include<stdio.h>
#include<conio.h>
void main()
{
 clrscr();
 int i;
 for(i=0;i<=2;i++)
{
   printf(“hello\n”);
}
getch() ;
}
```

```
\\output : hello
               hello
               hello
```

Program : write a program for 1 ,3,6,10,15

```
#include<stdio.h>
#include<conio.h>
void main()
{
 clrscr();
 int i, sum=0;
 for(i=1;i<=5;i++)
{
 sum = sum+ i;
 printf(“%d” ,sum);
 getch();
}
```

Output : 1,3,6,10,15

program : write a program for factorial number.

```
 #include <stdio.h>
#include <conio.h>
void main()
{
  int  i, fact=1;
  for(i=1; i<=5; i++)
{
  fact= fact*i;
}
printf(“%d”, fact);
getch();
}
```

\\output: 120

Program : write a program for febonacci series.

  0   1   1   2   3   5   8

Febonacci series means last two digit addition.

 0+1=1, 1+1=2, 2+1=3, 2+3=5, 3+5=8;

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

```
clrscr();

int  i, a=0, b=1, c;

for(i=1;i<=8;i++)

{

printf(“%d”, a);

c= a+b;

a=b;

b=c;

}

getch();

}

\\output : 0  1  1  2  3  5  8  13
```

Program : write a program find Armstrong number.

Armstrong number = 153

```
   153= 1*1*1+ 5*5*5+3*3*3

          =1+125+27

           =153

370= 3^3+7^3+0^3

      =27+343+0

       =370
```

Program

```
#include<stdio.h>
#include<conio.h>
void main ()
{
clrscr();
int  a, sum=0, x , n;
scanf ("%d", &a);
n=a;
for (sum=0 ; a>0 ; a/10)
{
   x=a%10;
sum= sum + x*x*x;
}
If (sum= =n)
{
printf("Arm no %d", n);
}
else
{
Printf ("Not an  Arm  no" ,n);
}
getch();
}
```

\\output : a=153 => Armstrong no

A=159 => Not an Armstrong no

Program : write a program for Palindrome number.

Palindrome number =212 (reverse)

212 (same)

Program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int n ,sum , x ,p;
p=n;
scanf("%d", &n);
for(sum=0;n>0;n=n/10)
{
x=n%10;
sum=sum*10+x;
}
if(sum= =p)
{
 printf("Palindrome no..");
}
else
{
 printf("Not a Palindrome no..");
}

getch();
}
```

  \\output : n=212=> palindrome no..

N=213=> not a Palindrome no..

Program : write a program for prime number.

```
#include<stdio.h>

#include<conio.h>

void main()

{

  clrscr();

  int i,n;

scanf("%d", &n);

for(i=2; i<n; i++)

{

if(n%i==0)

{

printf("not prime no.");

break;

}

}

if(n==i)

{
```

```
printf("prime no.");

}

getch();

}
```

\\output : 7=> prime no.

Problem: write a program for perfect number.

Perfect no. means all divisible by no. is sum is equal to this no.

Ex: 28= 1+2+4+7+14

=28

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  int  i, sum=0,n=28;
  for (i=1; i<n ; i++)
  if (n%i==0)
{
sum=sum+i;
}
if(sum==n)
{
printf("perfect no");
}
else
{
printf("not perfect no");

}
getch();

}
```

```
\\output : 28     perfect no

              5      not perfect no
```

Nested “ for ” loop :

C programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax :

```
for (initial; condition; increment/decrement )

{

for(initial; condition ; increment/decrement)

{

 Statement ;

}

Statement

}
```

Program : write a program

```
 *      *       *

 *      *      *

 *      *       *
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p;
for (i=1;i<=3;i++)
{
for(p=1;p<=3;p++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

```
*

*       *

*       *        *
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p;
for (i=1;i<=3;i++)
{
for(p=1;p<=i;p++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program to print

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p;
for (i=4;i>=1;i--)
{
for(p=1;p<=i;p++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p;
for (i=1;i<=4;i++)
{
for(p=1;p<=i;p++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p,k;
for (i=1;i<=4;i++)
{
for(p=4;p>i; p--)
{
printf(" ");
}
for(k=1;k<=i; k++)
{
printf("*");
}
printf("\n") ;
}
getch();    }
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int  i, p ,k, j;
for (i=1;i<=4;i++)
{
for(k=4;k>i; k--)
{
printf(" ");
}
for(p=1; p<=i; p++)
{
printf("*");
}
for(j=2;j<=i; j++)
{
printf("*");
}
printf("\n") ;
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
int i ,p ,k ,j;
for (i=1;i<=4;i++)
{
for(k=1;k<i; k++)
{
```

```
printf(" ");
}
for(p=4;p>=i;p--)
{
printf("*");
}
for(j=4;j>i; j--)
{
printf("*");
}
printf("\n") ;
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,p,k,j,m,n,l,q;
for (i=1;i<=4;i++)
{
for(k=1;k<i;k++)
{
printf(" ");
}
for(p=4;p>=i;p--)
{
printf("*");
}
for(j=4;j>i;j--)
{
printf("*");
}
printf("\n");
}

for(m=2;m<=4;m++)
{
for(n=4;n>m ;n--)
{
printf(" ");
}
for(l=1;l<=m;l++)
{
printf("*");
}
for(q=1;q<m;q++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,k,j,q,r;
for (i=1;i<=4;i++)
{
for(j=4;j>=i;j--)
{
printf("*");
}
for(k=1;k<i;k++)
{
printf(" ");
}
for(q=1;q<i;q++)
{
printf(" ");
}
for(r=4;r>=i;r--)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,k,j,q,r,p;
for (i=1;i<=4;i++)
{
for(j=4;j>=i;j--)
{
printf("*");
}
for(k=1;k<i;k++)
{
printf(" ");
}
for(q=1;q<i;q++)
{
printf(" ");
}
for(r=4;r>=i;r--)
{
printf("*");
}
printf("\n");
}
int s,t,u,v,w;
for(s=2;s<=4;s++)
{
for(t=1;t<=s;t++)
{
printf("*");
}
for(u=4;u>s;u--)
{
printf(" ");
}
for(v=4;v>s;v--)
{
printf(" ");
}
for(w=1;w<=s;w++)
{
printf("*");
}
printf("\n");
}
getch();
}
```

Program : write a program

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

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

Program: write a program
1
12
123
1234
12345

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

Program : write a program
1
2 4
3 6 9
4 8 12 16

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,k,m;
for (i=1;i<=4;i++)
{
for(k=1;k<=i;k++)
{
m=k*i;
printf("%d",m);
}
printf("\n");
}
getch();
}
```

Program : write a program

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

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

Program : write a program

```
 A
B C
D E F
G H I J
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,k,p;
char a='A';
for (i=1;i<=5;i++)
{
for(k=1;k<=i;k++)
{
printf("%c",a);
a++;
}
printf("\n");
}
getch();
}
```

Program

```
*1
**2
***3
****4
*****5
******6
*******7
```

```
int i,j,k,l,p=1;
            for(i=1;i<8;i++)
            {
                  for(j=1;j<=i;j++)
                  {
                        printf("*");
                  }
                  for(j=1;j<=i;j++)
                  {
                        if(i==p)
                        {
                        print(i);
                        p++;
                  }  }
                  printf("\n");
            }
      }   }
```

2. while loop :

A **while** loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.

Syntax :

Initialize

While (condition)

{

Body

}

Increment/decrement;

Problem : write a program to display Hello in 10 times .

```
#include <stdio.h>

#include<conio.h>

void main()

{

clrscr();

int i=1;

while(i<10)

{

printf(“Hello\n”);

i++;

}

getch();

}
```

```
\\output : Hello

                Hello

               Hello

               Hello

               Hello

               Hello

               Hello

               Hello

              Hello
```

Program: write a program to display the series .

1,3,6,10,15.

```
#include<stdio.h>
#include<conio.h>
void main()
 {
clrscr();
int i=1, k=0;
while (i<6)
{
k=i+k;
printf("%d\n"k);
i++;
}
getch();
}
```

```
\\output :   1
              3
             6
             10
             15
```

Program : write a program print this

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

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

Program : write a program

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i=1,j;
while(i<4)
{
j=1;
while(j<=i)
{
printf("*");
j++;
}
i++;
printf("\n");
}
getch();
}
```

\\output :

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

Program : write a program for display Armstrong number in 1 to 1000.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int n=1,p,k,sum,t;
while(n<=1000)
{
p=n;
for(sum=0;p>0;p=p/10)
{
k=p%10;
sum=sum+k*k*k;
}
if(n==sum)
{
printf("Armstrong no %d\n",sum);
}
n++;
}
getch();
}
```

2. while loop :

A **while** loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.

Syntax :

```
Initialize

While (condition)

{

Body

}

Increment/decrement;
```

Problem : write a program to display Hellow in 10 times .

```
#include <stdio.h>

#include<conio.h>

void main()

{

clrscr();

int i=1;

while(i<10)

{

printf("Hellow\n");

i++;

}

getch();

}
```

\\output : Hellow

Hellow

Hellow

```
Hellow

Hellow

Hellow

Hellow

Hellow

Hellow
```

Program: write a program to display the series .

1,3,6,10,15.

```
#include<stdio.h>
#include<conio.h>
void main()     {
clrscr();
int i=1, k=0;
while (i<6)
{
k=i+k;
printf("%d\n"k);
i++;
}
getch();
}
```

```
\\output :  1
            3
             6
             10
             15
```

Program : write a program to print

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

“do while” loop:

Unlike for and while loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming language checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax :

```
Do
Statement;
}
While (condition);
```

Program : write a program to display 1 to 10 number

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=1;
do
{
printf("%d\n",a);
a++;
}
while(a<=10);
getch();
}
```

Array

C programming language provides a data structure called “**the array”.**

Array of collection of similar data type and multiple data type in sequence form.

Syntax :

Type of Array

- Single Dimensional Array
- Double Dimensional Array
- Single Character Array

Type array name [array size];

- Why does the indexing start with zero in c?

Ans : In C, the name of an array is essentially a pointer, a reference to a memory location, and so the expression array[n] refers to a memory location n-elements away from the starting element. This means that the index is used as an offset. The first element of the array is exactly contained in the memory location that array refers (0 elements away), so it should be denoted as array[0].

Problem : write a program in 5 similar data is display.

```
#include<stdio.h>

#include<conio.h>

void  main()

{

clrscr();

printf (“%d %d %d %d %d” a[0],a[1],a[2],a[3],a[4]);

getch();

}

\\output : 1 2 3 4 5
```

Problem : run time output program

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[5],i;
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]);
}
for (i=0;i<=4;i++)
{
printf("%d",a[i]);
}
getch();
}
```

\\output : **1**

2

3

4

5

12345

Program : write a program find maximum number of array.

30, 5, 90, 2,6

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[5],i,max=0;
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]);
}
for (i=0;i<=4;i++)
{
if(max<a[i])
{
```

```
max=a[i];
}
}
printf("max no. %d",max);
getch();
}
```

```
\\output : 30

5

90

2

6

Max no. 90
```

Program : write a program find the minimum no. of array

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[5],i,min;
min=a[0];
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]);
}
for (i=0;i<=4;i++)
{
if(min>a[i])
{
min=a[i];
}
}
printf("min no. %d",max);
getch();
}
```

```
\\output : 2

3
```

4

10

1

Min no. 1

Program : write a program ascending order.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[5];
int i,j, temp;
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]) ;
}
for(i=0;i<=4;i++)
{
for(j=i+1;j<=4;j++)
{
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
for(i=0;i<=4;i++)
{
printf("%d\n",a[i]);
}
getch();
}
```

\\output :

```
2

4

1
```

7

3

Ans: 1

2

3

4

7

Program : write a program in descending order

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[5];
int i,j, temp;
for(i=0;i<=4;i++)
{
scanf("%d",&a[i]) ;
}
for(i=0;i<=4;i++)
{
for(j=i+1;j<=4;j++)
{
if(a[i]<a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
for(i=0;i<=4;i++)
{
printf("%d\n",a[i]);
}
getch();
}
```

\\output :

3

4

2

1

6

Ans

6

4

3

2

1

Double sided array :

A two-dimensional array is, in essence, a list of one-dimensional arrays.

Syntax :

Type arrayname [x][y];

Program : write a program to display double sided array.

```
#include<stdio.h>

#include<conio.h>

void main()    {

clrscr();

int a[2][3]={{1,2,3},{1,2,3}};

printf("%d %d %d %d %d %d",a[0][0],a[0][1],a[0][2],a[1][0],a[1][1],a[1][2]);

getch();

}
```

\\output : 1 2 3   1 2 3

Program : write a program addition of two array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,j,a[2][3],b[2][3],c[2][3];
for(i=0;i<=1;i++)
{
for(j=0;j<=2;j++)
{
scanf("%d  %d",&a[i][j],&b[i][j]);
}
}
for(i=0;i<=1;i++)
{
for(j=0;j<=2;j++)
{
c[i][j]=a[i][j]+b[i][j];
printf("%d\t",c[i][j]);
}
printf("\n");
}
getch();
}
```

```
\\output :  1 1 1

             1 1 1

            1 1 1

            1 1 1

Ans : 2 2 2

      2 2 2
```

Program : write the program find multiplication of two array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,j,a[2][3],b[2][3],c[2][3];
for(i=0;i<=1;i++)
{
for(j=0;j<=2;j++)
{
scanf("%d  %d",&a[i][j],&b[i][j]);
}
}
for(i=0;i<=1;i++)
{
for(j=0;j<=2;j++)
{
c[i][j]=a[i][j]*b[i][j];
printf("%d\t",c[i][j]);
}
printf("\n");
}
getch();
}
```

\\output :

```
1 2 3

1 2 3

1 2 3

1 2 3

Ans: 2 3 6

     2 3 6
```

Program : write a program find a diagonal of sum in matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,j,a[2][3],sum=0;
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d  %d",&a[i][j]);
}
}
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
if(i==j);
{
sum=a[i][j]+sum;
}
}
}
printf("%d",sum);
getch();
}
```

\\output :

```
1 2 3 4

1 2 3 4

1 2 3 4

Ans: 12
```

STRING

String is a sequence of characters that is treated as a single data item and terminated by null character '\0' . Remember that C language does not support strings as a data type. A **String** is actually one-dimensional array of characters in C

The string is define as “%s”.

%s= collection of character.

Program : write a program to display the charcter.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char a[]={'a','b','c'};
printf("%s",a);
getch();
}
\\output : abc
```

Program : run time program

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[5];
scanf(“%s”,a);
printf(“%s”,a);
getch();
}

\\output : kavita
Ans kavita
```

Program : find the length of character word.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char a[20];
int length;
scanf("%s",a);
for (length=0;a[length]!='\0';length++)
{
}
printf("%d",length);
getch();
}
```

```
\\output : kavita
  Ans : 6
```

String function :

1.strlen() : calculate the length of string.
2.**strcpy():** copy the string to another string.
3.strcat (): Concatenates(joins) two strings **.**
4.strrev() : to reverse the string .
5.**strupr() :**converts string to upper case
6.**strlwr():** converts string to lower case.
7. **strcmp():** compare two string.

- **gets() and puts():**

  Functions gets() and puts() are two string functions to take string input from user and display string respectively

  - gets() is used to count the space.
  - puts() is used to display the character length.

Program : write a program to find the length of character with space.

```
#include<stdio.h>
 #include<conio.h>
 void main()
 {
 clrscr();
 char a[20];
 int length;
 gets(a);
 for(length=0;a[length]!='\0';length++)
 {
 }
 printf("%d\n",length);
 puts(a);
 getch();
 }
```

\\output : kavita nikam

Ans : 12

Kavita nikam

Problem : write a program to find the length using strlen.

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

clrscr();

char a[6];

int len;

scanf("%s",a);

len=strlen(a);

printf("%d\n",len);

puts(a);

getch();

}
```

\\output :

```
Kavita

Ans :  6

         Kavita
```

Program : write a program to compare the two sting using strcmp.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
char a[6],b[6];
int len;
gets(a);
gets(b);
if(strcmp(a,b)==0)
{
puts("both are same");
}
else if(strcmp(a,b)>0)
{
puts("a is greater than b");
}
else
{
puts("b is greater than a");
}
getch();
}
```

\\output : ka

Vit

Ans : b is greater than a

Program : write a program to display hellow and its sleep after 1 sec.

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
void main()
{
clrscr();
int i;
for(i=0;i<=4;i++)
{
printf(“hellow\n”);
sleep(1);
}
\\Output : hellow
           hellow
           hellow
           hellow
           hellow
```

program : write a program lOtus to convert LoTUS.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
char data[10];
int i;
scanf("%s",data);
for(i=0;data[i]!='\0';i++)
```

```
{
if(data[i]>=65&&data[i]<=90)
{

data[i]=data[i]+32;
}
else if(data[i]>=97&&data[i]<=122)
{
data[i]=data[i]-32;
}
}
printf("%s",data);
getch();
```

Function

-It is the set of instruction .

-A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions. You can divide up your code into separate functions.

-how to declare function :

Declare
↓
Body
↓
Define

- Call by value and call by reference:
  Call by value essentially means that a copy of the variable is passed into the function. The function does not modify the original variable if it writes to the function input variable. Pass by reference means that essentially the variable itself is passed (though the name may change). Writes to the function input variable apply also to the original variable, and no copy of the variable is made when calling the function.

Type of function :

1. Take something return something
2. Take nothing return something
3. Take something return nothing
4. Take nothing return nothing

1. Take something return something
   It's a average program .

Program : write a program to find the average number.

```
#include<stdio.h>
#include<conio.h>
Float avg(int, int, int);
void main()
{
clrscr();
int a=100,b=200,c=300;
float d;
d=avg(a,b,c) ;
printf("%f",d);
getch();
}
Float avg(int x, int y, int z)
{
float k;
k=(x+y+z)/3.0;
return k;
}
```

\\output : 200.000

2. Take nothing return something : (clrscr)

   Program : write a program to find the average number.

```
#include<stdio.h>
#include<conio.h>
float avg();
void main()
{
clrscr();
float d;
d=avg() ;
printf("%f",d);
getch();
}
float avg()

{
float k;
int x=100,y=200,z=300;
k=(x+y+z)/3.0;
return k;
}
```

\\output: 200.00000

3. Take something return nothing :(void)
   Void does not return any value.
4. Take nothing return nothing : (main)

Program : write a program to find the average number.

```
#include<stdio.h>
#include<conio.h>
float avg();
void main()
{
clrscr();
avg() ;
getch();
}
float avg()

{
float k;
int x=100,y=200,z=300;
k=(x+y+z)/3.0;
printf("%f",k) ;
}
```

\\output: 200.0000

program : write a program Armstrong using function

```
#include<stdio.h>
#include<conio.h>
int avg();
void main()
{
```

```
clrscr();
int sum, x=153;
sum=avg();
if(sum= =x)
{
printf("armstrong");
}
else
{
printf("not") ;
}
getch();
}
int avg()
{
float k;
int n=153;
int sum=0;
for(sum=0;n>0;n=n/10)
{
k=n%10;
sum=sum+k*k*k;
}
return sum;
}
```

\\output: Armstrong

Program : write a program by swapping using call by reference.

```
#include<stdio.h>
#include<conio.h>
void swap(int *,int *);
void main()
{
clrscr();
int a=100,b=200;
swap(&a,&b);
getch();
}
void swap(int *p,int *q)
{
int *r;
*r=*p;
*p=*q;
*q=*r;
printf("%d %d",*p,*q);      }
```

- **Recursive function :**
  It is function to call function itself.
  It is used to without loop.

Recursion is the process of repeating items in a self-similar way. Same applies in programming languages as well where if a programming allows you to call a function inside the same function that is called recursive call of the function .

Syntax:

```
void recursion()

{

Recursion();

}


void main()

{

Recursion();

}
```

- Advantages:

  Recursion is more elegant and requires few variables which make program clean. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type.

-Disadvantages:

  In other hand, it is hard to think the logic of a recursive function. It is also difficult to debug the code containing recursion.

Program : write a program to find the factorial number using recursive function.

```
#include<stdio.h>

#include<conio.h>

int avg();

void main()
```

```
{
int fact(int ) ;
clrscr();
int p, x=5;
p=fact(x);
printf("%d",p);
getch();
}
int fact(int k)
{
int t;
if(k= =1)
{
return 1;
}
else
{
t=k*fact(k-1);
}
return t;
}
```

\\output: 120

Pointer

-Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers

-A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address.

-Syntax:

Type   * variable no.;

Program : write a program to display value a.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100, *p;

p= &a;

printf("%d",*p);

getch();

}

\\output : 100
```

Program :

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100,*p, b=200, *q;

p=&a;
q=&b;

printf("%d %d %d %d",*p, a,*q, b);

getch();

}
```

\\output : 100 100 200 200

Program : to share some data in 4 variable.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100, *p,**q,***r,****s;

p=&a;

q=&p;

r=&q;

s=&r;

printf("%d %d %d %d",*p,**q,***r,****s);

getch();

}
```

\\output : 100 100 100 100

Program write program how to use for loop in a pointer

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

char k='T',*p;
for(p=&k;*p!='\0';*p++)
{
*p=*p+32;
}
printf("%c",k);
getch();
}
```

Program : write a program in swapping with 3rd variable by using pointer.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a=100,b=200,*p,*q,*temp;
p=&a;
q=&b;
*temp=*p;
*p=*q;
*q=*temp;
printf("%d %d",*p,*q);
getch();
}
```

\\output : 200 100

Program : write a program in swapping with 2 variable by using pointer.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100, b=200, *p, *q;

p=&a;

q=&b;

*p=*p+*q;

*q=*p-*q;

*p=*p-*q;

printf("%d %d",*p,*q);

getch();

}

\\output : 200 100
```

Program : write a program find a greater number using pointer.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int a=100, b=200, *p, *q;

p=&a;

q=&b;

if(*p>*q)

{

printf("a greater than b");
}
else
{
printf("b greater than a");
}
getch();
}
```

Program : write a program Find Largest Element by Largest Element

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int a[7],*p,max,i;
Printf(“Enter a Array Elements\n”);
for(i=0;i<=6;i++)
{
scanf(“%d”,&a[i]);
}
for(i=0;i<=6;i++)
{
*p=a[i];
if(*p>max)
{
max=*p;
}
}
printf(“Maximum Elements in Pointer %d”,max);
getch();      }
```

```
Maximum Element :780
```

\\output : b greater than a

Program : write a program to find the size in int, char, float, string, using “%u”.

```
#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

printf("%u",sizeof(45));

printf("\n%u",sizeof(345.678));

printf("\n%u",sizeof(34.56f));

printf("\n%u",sizeof('a'));

printf("\n%u",sizeof("as"));

getch();   }
```

\\output :

```
2

8

4

1

3
```

Structure

A struct in the c- programming language (and many derivative) is a complex data type declaration that defines a physically grouped list of variable to be placed under one name in a block of memory allowing the difference variable to be accessed via a single pointer or the struct declared name which return the same.

Syntax :

```
struct  struct name

{

Body (member define)

};
```

Program : write a program to display one student information.

```
#include<stdio.h>
#include<conio.h>
struct student
{
int id;
char name[10];
};
void main()
{
clrscr();
student s1;
printf("enter a data\n");
scanf("%d %s",&s1.id,s1.name);
printf("%d %s",s1.id,s1.name);

getch();
}
```

\\output : 23

Kavita

Ans: 23 kavita

Program : write a program to display many student data.

```
#include<stdio.h>

#include<conio.h>

struct student

{

int id;

char name[10];

};

void main()

{

clrscr();

student s1[4];

int i;

printf("enter a data\n");

for(i=0;i<=3;i++)

{

scanf("%d %s",&s1[i].id,s1[i].name);

}

for(i=0;i<=3;i++)

{

printf("%d %s\n",s1[i].id,s1[i].name);

}

getch();

}
```

\\output :

```
1

K

2
```

R

3

L

4

H

5

P

Ans : 1 K

2 R

3 L

4 H

5 p

#difference between the function and macros:

Keyword	function	macros
Memory required	Less or only one copy exists	More, since inline code is produced.
Time required	More since control shift to called "function"	Less due to inline expansion.
Data type	Considered by compiler for function invocation	Not considered since text replacement taken place before compilation.
Use	Implement a complex logic for given task	Small code.

Pre processor director

-The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation. We'll refer to the C Preprocessor as the CPP.

macros:

- the c pre processor is a tool that processor source code before it is compiled.

-macros are pre processor directive that are defined using “#define” derective.

-two types : 1) simple macros.

2) macro with arguments(function macros).

1) simple macros : macro with no arguments is called “simple macros. ”

-syntax :

#define MACRONAME macro substitution text

Program : to display hellow word in 5 times .

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

\\output:

```
hellow

hellow

hellow

hellow

hellow
```

program : write a program to addition of two number .

```
#define A+B
#define A 20
#define B 30
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("%d",A+B);
getch();
}
```

\\output : 50

2) function macro :

Program : to find the interest is given value.

```
#define SI(P,R,T) (P+R+T)/100
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
clrscr();
int P=100,R=100,T=200;
printf("%d",SI(P,R,T));
getch();
}
```

\\output : 4

Storage Class

-A data type of variable indicates the type of data that will be stored in the variable and the amount of memory to be allocated to the variable.

- 4 type : 1) automatic(auto)

2) static

3) global

4) register

1) automatic storage class:

- variable having automatic storage class are called “automatic” or “local variable”

-it is default storage class of variable declared without any specific storage classes.

-syntax :

```
    Auto int variableName;
```

2) static storage class :

-A variable declared with keyword static are called “static variable”

-syntax:

```
static int variableName;
```

-lifetime is till the program execute.

3) global storage class: (extern)

-initial value is zero

-external variables are defined outside any function . they are global to the entire program.

-lifetime of extern variable is until the end of program execution.

4) register storage class :

-storage is within CPU register.

-normally , when operation are carried out info is transferred from the memory to the registers. The result are then transferred back from register to the memory . this take some time variable like loop counter, are required number of times for faster execution of loops.

The loop counter variable can be stored in cpu register itself.

Syntax : Register int varaibleName;

Summary:

keyword	auto	register	static	global
scope	local	local	Within a function	Across file
life	Within block	Within block	Throughout program	Throughout program
Initial value	garbage	garbage	Zero	Zero
Memory	On stack	register	Data section	Data section

C++

Introduction to C++

C++, as we all know is an extension to C language and was developed by Bjarne stroustrup at bell labs. C++ is an intermediate level language, as it comprises a confirmation of both high level and low level language features. C++ is a statically typed, free form, multiparadigm, compiled general-purpose language.

C++ is an Object Oriented Programming language but is not purely Object Oriented. Its features like Friend and Virtual, violate some of the very important OOPS features, rendering this language unworthy of being called completely Object Oriented. Its a middle level language.

Benefits of C++ over C Language

The major difference being OOPS concept, C++ is an object oriented language whereas C language is a procedural language. Apart form this there are many other features of C++ which gives this language an upper hand on C language.

Following features of C++ makes it a stronger language than C

There is Stronger Type Checking in C++.

All the OOPS features in C++ like Abstraction, Encapsulation, Inheritance etc makes it more worthy and useful for programmers.

C++ supports and allows user defined operators (i.e Operator Overloading) and function overloading is also supported in it.

Exception Handling is there in C++.

The Concept of Virtual functions and also Constructors and Destructors for Objects.

Inline Functions in C++ instead of Macros in C language. Inline functions make complete function body act like Macro, safely.

Variables can be declared anywhere in the program in C++, but must be declared before they are used.

Difference between C and C++

S No.	C Language	C++ Language
1.	**Procedure** Oriented Language	**Object** Oriented Language
2.	**Middle** level Language	**High** level Language
3.	Header files: <**stdio.h**>	Header files: <**iostream.h**>
4.	It has print function : **printf()** It has input function : **scanf()**	It has print object : **cout** << It has input object : **cin** >>
5.	It has memory allocation function as : **malloc()** and Memory dellocation function as : **calloc()**	It has memory allocation function as : **constructor** Memory dellocation function as : **destructor**
6.	It has **pointer(*)** data type	It does not have **pointer** data type
7.	It does not have access specifiers	It has access specifiers: **public** , **protected** and **private**
8.	C does not have **friend** function	C++ have **friend** function
9.	C does not have **template**	C++ have **template**

Namespaces

Consider a situation, when we have two persons with the same name, Zara, in the same class. Whenever we need to differentiate them definitely we would have to use some additional information along with their name, like either the area if they live in different area or their mother or father name, etc.

Same situation can arise in your C++ applications. For example, you might be writing some code that has a function called xyz() and there is another library available which is also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code.

A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope

Object oriented Languages

Object means a real word entity such as pen ,chair ,table

Object Oriented Programming is a methodology or paradigm a program using classes and objects It simplifies the software

Development and maintenance by providing some concepts

=>Objects

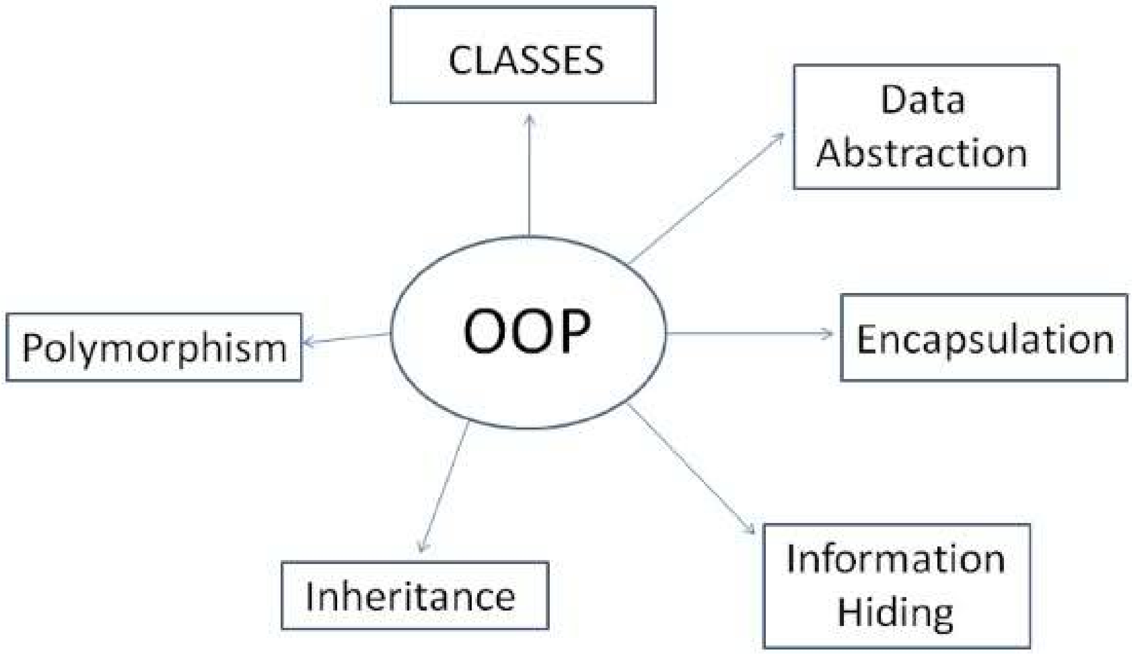
=>Class

=>inheritance

=>Polymorphism

=>Abstraction

=>Encapsulation



Objects:

Is a real time entity which is having attributes,behaviours ,Identity

That is a Object

Class :Collections of objects

Inheritance:

when a one class property acquired by another class that Inheritance

Type of Inheritance

1.Single Level Inheritance

2. Multiple Inheritance

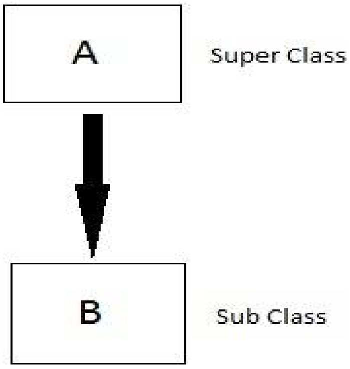
3.Multilevel Inheritance

4.Hybrid Inheritance

5.Hierachical Inheritance

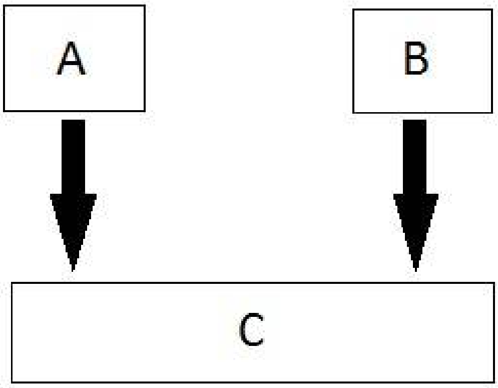
Single Level Inheritance

One class Properties acquired by another class

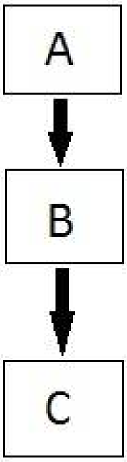


Multiple Inheritance

Single Derived class Inherits From two or more than two base classes

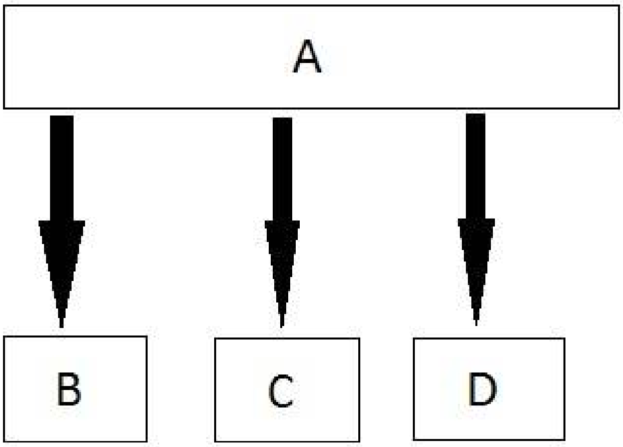


3. Multilevel Inheritance inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.

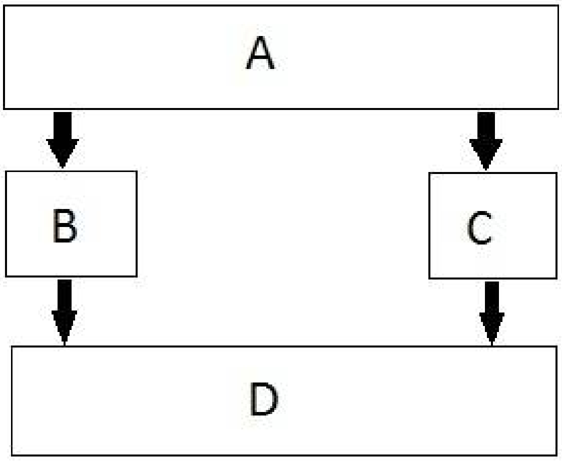


4.Hybrid Inheritance

Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.



5.Hierachical Inheritance : Multiple derived classes inherits from a single base class.



Polymorphism

One task perform by different different way that is called Polymorphism

Function Overriding

If we inherit a class into the derived class and provide a definition for one of the base class's function again inside the derived class, then that function is said to be **overridden**, and this mechanism is called **Function Overriding**

Requirements for Overriding

1. Inheritance should be there. Function overriding cannot be done within a class. For this we require a derived class and a base class.
2. Function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

Connecting the function call to the function body is called **Binding**. When it is done before the program is run, its called **Early** Binding or **Static** Binding or **Compile-time** Binding.

Abstraction

Hiding the unusual data and showing use full data that is called

Abstraction

Encapsulation

Combination of class, variable and method and Encapsulation

Access Specifier

Access	**public**	**protected**	**private**
Same class	Yes	Yes	Yes
Derived classes	Yes	Yes	No
Outside classes	Yes	No	No

Variable

Variable is memory allocation which is stored some data that is a variable

Type of Variable

1. **Local variable**
2. **Class variable**
3. **Static variable**

=>Local variable

if a variable declare inside of method or block or constructor that is variable

- ⇨ **Class Variable or Global   Variable**
- ⇨ if a variable declare inside of class that is a class variable
- ⇨ Static variable
- ⇨ Static is properties which are properties common for al objects
  That is Static Variable

Write a program make object and class as well as method

```
#include<iostream.h>
#include<conio.h>
class Student{
public:
void get()
{
cout<<"Hello C/C++";
}
};
void main()
{
clrscr();
Student S1;
S1.get();
getch();
}
```

output => Hello C/C++

Write a program make object and, class Variable class as well as method

```
#include<iostream.h>
#include<conio.h>
class Student{
public:
int a;//class variable
void get()
{
a=1000;
cout<<"Hello C/C++  "<<a;
}
};
void main()
{
clrscr();
Student S1;
S1.get();
getch();
}
output => Hello C/C++ 1000
```

Write a program make object and,class Variable, and local variable class as well as method

```
#include<iostream.h>
#include<conio.h>
class Student{
public:
int a;//class variable
void get()
{
int b,c;//local variable
b=100;
c=100;
a=b+c;
}
void show()
{
cout<<a;
}
};
void main()
```

```
{
clrscr();
Student S1;
S1.get();
S1.show();
getch();
}
```

output => Addition of two number 200

Write a code create multiple objects in class

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class Student{
⇨ public:
⇨ int a;//class variable
⇨ void get()
⇨ {
⇨ int b,c;//local variable
⇨ b=100;
⇨ c=100;
⇨ a=b+c;
⇨ }
⇨ void show()
⇨ {
⇨ cout<<a<<"\n";
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Student S1,S2,S3;
⇨ S1.get();
⇨ S1.show();
⇨ S2.get();
⇨ S2.show();
⇨ S3.get();
⇨ S3.show();
  getch();}
```

```
⇨ output => Addition of two number 200
⇨           Addition of two number 200
⇨           Addition of two number 200
```

Write a program make Parameters method

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class Student{
⇨ public:
⇨ int a;//class variable
⇨ void get(int b,int c)
⇨ {
⇨ a=b+c;
⇨ }
⇨ void show()
⇨ {
⇨ cout<<a<<"\n";
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Student S1;
⇨ S1.get();
⇨ S1.show();
  getch();}
```

⇨ **output => Addition of two number 200**

How to create multiple classes in programs

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class Student{
⇨ public:
⇨ void get()
⇨ {
⇨ cout<<"Hello C";
⇨ }
⇨ };
⇨ class Demo{
⇨ public:
⇨ void put()
⇨ {
⇨ cout<<"C++";
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Student S1;
⇨ Demo D1;
⇨ S1.get();
⇨ D1.put();
⇨ getch();
⇨ }
```

```
⇨ output => C
⇨            C++
```

Write a program copy data from local variable to class variable

```
class Student   {
private :
int a;
char b[10];
//class variable
public:
//private:
void get(int  c,char d[]) //parameterized
{
//int a;
a=c;
strcpy(b,d);
}
void put()
{
 cout<<b<<" : "<<a;
}
};
void main()
{
clrscr();
Student S1;
S1.get(100,"Hello");
S1.put();
getch();     }
```

Write a program for method overloading

//Method overloading

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Student{
private :
int a;
char b[10];
//class variable
public:
//private:
void get(int c,char d[]) //parameterized
{
//int a;
a=c;
strcpy(b,d);
}
void get(int h)
{
 cout<<b<<" : "<<a<<h;
}
void get()
{
cout<<"\nWelcome to method overloading!";
}
};
void main()
{
clrscr();
Student S1;
S1.get(100,"Hello");
S1.get(200);
S1.get();
getch();
}
```

Write a program make single level inheritance

```
#include<iostream.h>
#include<conio.h>
class Cmp{
private :
char cname[10];
protected :
void get()
{
cout<<"Enter a Company Details\n";
cin>>cname;
}
void show()
{
cout<<cname;
}
};
class Emp : Cmp{
char ename[10];
public:
void put()
{
Cmp::get();
cout<<"Enter a employee Details\n";
cin>>ename;
}
void display()
{
Cmp::show();
cout<<ename;
}
};
```

```
void main()
{
clrscr();
Emp E1;
E1.put();
E1.display();
getch();
}
```

Output =>

```
Enter a Company Name
Wipro
Enter a Employee Name
Baba
Wipro
Baba
```

Write a program use of access Specifier Protected

```
#include<iostream.h>
#include<conio.h>
class Cmp{
private :
char cname[10];
protected :
void get()
{
```

```
cout<<"Enter a Company Details\n";

cin>>cname;

}

void show()

{

cout<<cname;

}

};

class Emp : Cmp{

char ename[10];

public:

void get()

{

Cmp::get();

cout<<"Enter a employee Details\n";

cin>>ename;

}

void show()

{

Cmp::show();

cout<<ename;

}

};

void main()

{

clrscr();
```

```
Emp E1;

E1.get();

E1.show();

getch();

}
```

- **Output =>**
- Enter a Company Name
- Wipro
- Enter a Employee Name
- Baba
- Wipro
- Baba

Method Overloading

- **In a same class method name parameters list different that**
- **Is a method overloading**
- #include<iostream.h>
- #include<conio.h>
- class Demo{
- public :
- void get()
- {
- cout<<"Mrthod Overload!!!!!!\n" ;
- }
- void get(int a)
- {
- cout<<"Method Overloading$$$$\n" ;
- }
- void get(int b,int a)
- {
- cout<<"Method Overloading%%%\n";
- }
- };
- void main()
- {
- clrscr();
- Demo D1;
- D1.get();
- D1.get(123);
- D1.get(123,56);

```
⇨ getch();
⇨ }
⇨ Output : Mrthod Overload!!!!!!
⇨               Method Overloading$$$$
⇨              Method Overloading%%%
```

Method Overriding

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class IT{
⇨ public:
⇨ void get(int a)
⇨ {
⇨  cout<<"hi \n";
⇨  }
⇨  };
⇨
⇨   class Subcomp      :IT{
⇨   public:
⇨   void get(int a)
⇨   {
⇨   IT::get(789);
⇨      cout<<"user\n";
⇨     }
⇨     };
⇨     void main()
⇨     {
⇨     clrscr();
⇨     Subcomp S1;
⇨  //   S1.get();
⇨     S1.get(11);
⇨     getch( );
⇨ }
```

Virtual Function

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class Shape{
⇨ private :
⇨ public :
⇨ int heigth,width;
⇨ public :
⇨ void get()
⇨ {
⇨ cout<<"Enter a width & heigth \n";
⇨ cin>>heigth>>width;
⇨ }
⇨ virtual void area(){};
⇨ };
⇨ class Tri :public Shape {
⇨ public :
⇨ void area()
⇨ {
⇨ float tl1=.2*heigth*width;
⇨ cout<<"Tringle of area " <<tl1;
⇨ }
⇨ };
⇨ class Rect :public Shape{
⇨ public:
⇨ void area()
⇨ {
⇨ float tl2=heigth* width;
⇨ cout<<"Rectangle of area "<<tl2;
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Shape *s1,*s2;
⇨ Tri t1;
⇨ Rect r1;
⇨ s1=&t1;
⇨ //s1->t1;
⇨ s1->get();
⇨ s1->area();
⇨ s2=&t1;
⇨ s2->get();
⇨ s2->area();
⇨ getch();
⇨ }
```

Virtual Function

```
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ class Classes{
⇨ public :
⇨ int fees;
⇨ void get()
⇨ {
⇨  cout<<"Enter a Fees\n";
⇨  cin>>fees;
⇨ }
⇨ virtual void languages(){};
⇨ };
⇨ class C : public Classes{
⇨ public :
⇨ void languages()
⇨ {
⇨ int tl1=2*fees;
⇨ cout<<"Total Fees of C "<<tl1<<"\n";
⇨ }
⇨ };
⇨ class Java : public Classes{
⇨ void languages()
⇨ {
⇨ int tl2=4*fees;
⇨ cout<<"Total Fees of java "<<tl2;
⇨
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Classes *C1,*C2;
⇨ C c;
⇨ Java j;
⇨ C1=&c;
⇨ C2=&j;
⇨ C1->get();
⇨  C2->get();
⇨  C1->languages();
⇨
   C2->languages();
⇨  getch();     }
```

Operator loading

```
⇨  #include<iostream.h>
⇨  #include<conio.h>
⇨  class Demo{
⇨  public :
⇨  int a;
⇨  int b;
⇨  public :
⇨  void get()
⇨  {cout<<"Enter a values \n";
⇨  cin>>a>>b;
⇨  }
⇨  void display()
⇨  {
⇨   cout<<a<<" : "<<b<<"\n";
⇨  }
⇨  void operator ++()
⇨  {
⇨   a++;
⇨   b++;
⇨  }
⇨  void operator ++(int a)
⇨  {
⇨  a++;
⇨  b++;
⇨  }
⇨  };
⇨  void main()
⇨  {
⇨  clrscr();
⇨  Demo D1,D2,D3;
⇨  D1.get();
⇨  D2.get();
⇨  D3.a=D1.a+D2.a;//100//100=200
⇨  D3.b=D1.b+D2.b;//200//200=400
⇨  D1.display();
```

- ⇨ D2.display();
- ⇨ D3.display();
- ⇨ getch();
- ⇨ }

- ⇨ **Data abstraction** refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- ⇨ Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.
- ⇨ Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.
- ⇨ Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals.
- ⇨ Now, if we talk in terms of C++ Programming, C++ classes provides great level of **data abstraction**. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.
- ⇨ For example, your program can make a call to the **sort()** function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class Adder{
   public :
             void addNum(int number)
             {
               total=0;
               total+=number;
               }
               int getTotal()
               {
                 return total;
               };
               private :
               int total;
               };
               int main()
               {
                 Adder a;

                 a.addNum(300);
                 cout<<"Total  "<<a.getTotal()<<endl;
                 getch();
                 return 0;
                 }
```

Data encapsulation

All C++ programs are composed of the following two fundamental elements:

- **Program statements (code):** This is the part of a program that performs actions and they are called functions.
- **Program data:** The data is the information of the program which affected by the program functions.

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of **data hiding**.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called **classes**. We already have studied that a class can contain **private, protected** and **public** members.

By default, all items defined in a class are private. For example:

```
#include <iostream.h>

#include<conio.h>

using namespace std;

class Adder{

   public:

      // constructor

      Adder(int i = 0)

{

         total = i;

      }

        // interface to outside world

      void addNum(int number) {

         total += number;

      }

          // interface to outside world

      int getTotal() {

         return total;

      };

          private:

      // hidden data from outside world

      int total;
};

int main( )

{

   Adder a(1);

   a.addNum(10);

   a.addNum(20);

   a.addNum(30);

   cout << "Total " << a.getTotal() <<endl;

   getch();

   return 0;

}
```

Template is generic which offer user defined data type

```
⇨ Template
⇨
⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ template<class P,class Q>
⇨
⇨ void get(P p,Q q)
⇨ {
⇨ cout<<p<<"  :  "<<q<<"\n";
⇨ q=p+q;
⇨ cout<<q<<"\n";
⇨
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ get(12,34);
⇨ get(12.56,45.67);
⇨ get('a','b');
⇨ getch();
⇨ }

⇨ #include<iostream.h>
⇨ #include<conio.h>
⇨ template<class T1,class T2>
⇨ class Template{
⇨ private :
⇨
⇨ T1 a,b;
⇨ T2 c;
⇨ public :
⇨ Template(T1 x,T2 y)
⇨ {
⇨ a=x;
⇨ c=y;
⇨ b=a+c;
⇨ cout<<" Addition : "<<b;
⇨ }
⇨ };
⇨ void main()
⇨ {
⇨ clrscr();
⇨ Template<int,int>(120,80);
```

⇨ Template<float,float>(122.5,76.44);
⇨ getch();
⇨ }

What is a virtual base class?

- An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.
- C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

What is Virtual base class? Explain its uses.

- When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word virtual.
- Consider the following example :

```
class A
{
  public:
      int i;
};

class B : virtual public A
{
  public:
      int j;
};

class C: virtual public A
{
  public:
      int k;
};

class D: public B, public C
```

```
{
   public:
       int sum;
};1

int main()
{
   D ob;
   ob.i = 10; //unambiguous since only one copy of i is inherited.
   ob.j = 20;
   ob.k = 30;
   ob.sum = ob.i + ob.j + ob.k;
   cout << “Value of i is : ”<< ob.i<<”\n”;
   cout << “Value of j is : ”<< ob.j<<”\n”; cout << “Value of k is :”<< ob.k<<”\n”;
   cout << “Sum is : ”<< ob.sum <<”\n”;

   return 0;
}
```

File Handling

C++ provides the following classes to perform output and input of characters to/from files:

- **ofstream:** Stream class to write on files
- **ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes `istream` and `ostream`. We have already used objects whose types were these classes: `cin` is an object of class `istream` and `cout` is an object of class `ostream`. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to us e `cin` and `cout`, with the only difference that we have to associate these streams with physical files. Let's see an example:

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

```
[file example.txt]
Writing this to a file.
```

Edit
&
Run

This code creates a file called `example.txt` and inserts a sentence into it in the same way we are used to do with `cout`, but using the file stream `myfile` instead.

But let's go step by step:

Open a file
The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a *stream* (i.e., an object of one of these classes; in the previous example, this was `myfile`) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function `open`:

```
open (filename, mode);
```

Where **`filename`** is a string representing the name of the file to be opened, and `mode` is an optional parameter with a combination of the following flags:

`ios::in`	Open for input operations.
`ios::out`	Open for output operations.
`ios::binary`	Open in binary mode.
`ios::ate`	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
`ios::app`	All output operations are performed at the end of the file, appending the content to the current content of the file.
`ios::trunc`	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

All these flags can be combined using the bitwise operator OR (`|`). For example, if we want to open the file `example.bin` in binary mode to add data we could do it by the following call to member function `open`:

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each of the `open` member functions of classes `ofstream`, `ifstream` and `fstream` has a default mode that is used if the file is opened without a second argument:

Class	default mode parameter
`ofstream`	ios::out
`ifstream`	ios::in
`fstream`	ios::in | ios::out

For **`ifstream`** and **`ofstream`** classes, **`ios::in`** and **`ios::out`** are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the `open` member function (the flags are combined).

For **`fstream`**, the default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

File streams opened in *binary mode* perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

Since the first task that is performed on a file stream is generally to open a file, these three classes include a constructor that automatically calls the `open` member function and has the exact same parameters as this member. Therefore, we could also have declared the previous `myfile` object and conduct the same opening operation in our previous example by writing:

```
ofstream myfile ("example.bin", ios::out | ios::app |
ios::binary);
```

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.

To check if a file stream was successful opening a file, you can do it by calling to member `is_open`. This member function returns a `bool` value of `true` in the case that indeed the stream object is associated with an open file, or `false` otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Closing a file

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function `close`. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.

In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function `close`.

Text files

Text file streams are those where the `ios::binary` flag is not included in their opening mode. These files are designed to store text and thus all values that are input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Writing operations on text files are performed in the same way we operated with `cout`:

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another
line.\n";
    myfile.close();
  }
  else cout << "Unable to open
file";
  return 0;
}
```

```
[file example.txt]
This is a line.
This is another line.
```

Edit & Run

Reading from a file can also be performed in the same way that we did with `cin`:

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( getline (myfile,line)
)
    {
      cout << line << '\n';
    }
    myfile.close();
  }

  else cout << "Unable to open
file";

  return 0;
}
```

```
This is a line.
This is another line.
```

Edit & Run

This last example reads a text file and prints out its content on the screen. We have created a while loop that reads the file line by line, using `getline`. The value returned by `getline` is a reference to the stream object itself, which when evaluated as a boolean expression (as in this while-loop) is `true` if the stream is ready for more operations, and `false` if either the end of the file has been reached or if some other error occurred.

Operator overloading

It is a type of polymorphism in which an **operator** is**overloaded** to give user **defined** meaning to it.**Overloaded operator** is used to perform operation on user-**defined** data type. For examp le '+' **operator** can be **overloaded** to perform addition on various data types, like for Integer, String(concatenation) etc

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
struct String{
char str[20];
};
int operator ==(String s1,String s2)
{
 if(strcmp(s1.str,s2.str)==0)
 {
  return 1;
 }
 else
 {
  return 0;
 }
}
int main()
{
clrscr();
String str1,str2;
cout<<"\nInput String11 :";
cin>>str1.str;
cout<<"\n Input String22 :" ;
cin>>str2.str;
if(str1==str2)
{
cout<<" String is Same ";
}
else
{cout<<"Not is Same";
}
getch();
}
```

```
#include<iostream.h>
#include<conio.h>
class date{
int dd,mm,yy;
public:
void read_date()
{
cout<<"Input date :";
cin>>dd>>mm>>yy;
}
int operator ==(date d)
{
if(dd==d.dd &&mm==d.mm&&d.yy&&yy==d.yy)
{

return 1;
}
else
{
 return 0;
}
}
};
void main()
{
clrscr();
date date1,date2;
cout<<"\nInput date1 :=";
date1.read_date();
cout<<"\nInput date2 :=";
date2.read_date();
if(date1==date2)
{
cout<<"Equals";
}
else
{
cout<<"Not Equals";
}
getch();
}
```

Const Keyword

Constant is something that doesn't change. In C and C++ we use the keyword const to make program elements constant. Const keyword can be used in many context in a C++ program. Const keyword can be used with:

Variables
Pointers
Class Member functions
Objects

1) Constant Variables

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while declared.

```
int main
{
 const int i = 10;
 const int j = i+10;  // Works fine
 i++;    // This leads to Compile time error
}
```

In this program we have made i as constant, hence if we try to change its value, compile time error is given. Though we can use it for substitution.

```
#define MAX(num1,num2) (num1>num2?num1:num2)
#include<stdio.h>
#include<conio.h>
V5oid main()
{
clrscr();
int res,no1=200,no2=24;
int result,f1,f2;
res=MAX(no1,no2);
result=MAX(f1,f2);
printf("%d %d",res,result);
getch();
}

#define SI(p,n,r) p*n*r/100
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
float p=1000,n=1,r=8.8;
printf("%f",SI(p,n,r));
getch();
}
```

1. malloc()

The name malloc stands for "memory allocation". The function **malloc()** reserves a block of memory of specified size and return a pointer of type **void** which can be casted into pointer of any form.

Syntax of malloc()

```
ptr=(cast-type*)malloc(byte-size)
```

Here, **ptr** is pointer of cast-type. The **malloc()** function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

```
ptr=(int*)malloc(100*sizeof(int));
```

This statement will allocate either 200 or 400 according to size of **int** 2 or 4 bytes respectively and the pointer points to the address of first byte of memory.

2. calloc()

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax of calloc()

```
ptr=(cast-type*)calloc(n,element-size);
```

This statement will allocate contiguous space in memory for an array of **n** elements. For example:

```
ptr=(float*)calloc(25,sizeof(float));
```

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

3. free()

Dynamically allocated memory with either calloc() or malloc() does not get return on its own. The programmer must use free() explicitly to release space.

Syntax of free()

```
free(ptr);
```

This statement cause the space in memory pointer by ptr to be deallocated.

Examples of calloc() and malloc()

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
    if(ptr==NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using calloc() function.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));
    if(ptr==NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

Difference Between malloc and calloc

Differences between malloc and calloc

malloc	calloc
The name malloc stands for *memory allocation*.	The name calloc stands for *contiguous allocation*.
void *malloc(size_t n) returns a pointer to n bytes of uninitialized storage, or NULL if the request cannot be satisfied. If the space assigned by malloc() is overrun, the results are undefined.	void *calloc(size_t n, size_t size)returns a pointer to enough free space for an array of n objects of the specified size, or NULL if the request cannot be satisfied. The storage is initialized to zero.
malloc() takes one argument that is,*number of bytes*.	calloc() take two arguments those are:*number of blocks* and *size of each block*.

syntax of malloc():
void *malloc(size_t n);
Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. OtherwiseNULL is returned.
	syntax of calloc():
	void *calloc(size_t n, size_t size);
	Allocates a contiguous block of memory large enough to hold n elements of sizebytes each. The allocated region is initialized to zero.

malloc is faster than calloc.	calloc takes little longer than mallocbecause of the extra step of initializing the allocated memory by zero. However, in practice the difference in speed is very tiny and not recognizable.

# and ## Operators in C

Stringizing operator (#)

This operator causes the corresponding actual argument to be enclosed in double quotation marks.

The # operator, which is generally called the stringize operator, turns the argument it precedes into a quoted string.

For more on pre-processor directives – refer this

Examples :

```
#include <stdio.h>

#define mkstr(t) #t

int main(void)

{

   printf(mkstr(lotusithub));

   printf(mkstr(welcome));

   return 0;

}
```

Allows tokens used as actual arguments to be concatenated to form other tokens.

It is often useful to merge two tokens into one while expanding macros.

This is called token pasting or token concatenation.

The ‘##’ pre-processing operator performs token pasting.

When a macro is expanded, the two tokens on either side of each ‘##’ operator are combined into a single token, which then replaces the ‘##’ and the two original tokens in the macro expansion.

Example :

The preprocessor transforms printf("%d", concat(x, y)); into printf("%d", xy);

// CPP program to illustrate (##) operator

```
#include <stdio.h>

#define concat(a, b) a##b

int main(void)

{

   int xy = 30;

   printf("%d", concat(x, y));

   return 0;

}
```

1. Automatic Storage Class

A variable defined within a function or block with auto specifier belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block.

```
#include <stdio.h>

int main( )

{

 auto int i = 1;

 {

  auto int i = 2;

  {

   auto int i = 3;

   printf ( "\n%d ", i);

  }

  printf ( "%d ", i);

 }

 printf( "%d\n", i);

}
```

2. Register Storage Class

The register specifier declares a variable of register storage class. Variables belonging to register storage class are

local to the block which they are defined in, and get destroyed on exit from the block. A register declaration is equivalent to an auto declaration, but hints that the declared variable will be accessed frequently; therefore they are placed in CPU registers, not in memory. Only a few variables are actually placed into registers, and only certain types are eligible; the restrictions are implementation-dependent. However, if a variable is declared register, the unary & (address of) operator

may not be applied to it, explicitly or implicitly. Register variables are also given no initial value by the compiler.

```
#include <stdio.h>

int main()

{

 register int i = 10;

 int *p = &i; //error: address of register variable requested

 printf("Value of i: %d", *p);

 printf("Address of i: %u", p);

}
```

4. External Storage Class

The extern specifier gives the declared variable external storage class. The principal use of extern is to specify that a variable is declared with external linkage elsewhere in the program. To understand why this is important, it is necessary to understand the difference between a declaration and a definition. A declaration declares the name and type of a variable or function. A definition causes storage to be allocated for the variable or the body of the function to be defined.

The same variable or function may have many declarations, but there can be only one definition for that variable or function.When extern specifier is used with a variable declaration then no storage is allocated to that variable and it is assumed that the variable has already been defined elsewhere in the program. When we use extern specifier the variable cannot be initialized because with extern specifier variable is declared, not defined.

In the following sample C program if you remove extern int x; you will get an error "Undeclared identifier 'x'" because

variable x is defined later than it has been used in printf. In this example, the extern specifier tells the compiler that

variable x has already been defined and it is declared here for compiler's information.

```
#include <stdio.h>

extem int x;

int main()

{

  printf("x: %d\n", x);

}

int x = 1000;
```

Enumeration (or enum) is a user defined data type in C.

It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

```
#include<stdio.h>

 enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,

                    Aug, Sep, Oct, Nov, Dec};

 int main()

{

  int i;

  for (i=Jan; i<=Dec; i++)

     printf("%d ", i);

      return 0;

}
```

restrict keyword in C

In the C programming language (after 99 standard), a new keyword is introduced known as restrict. restrict keyword is mainly used in pointer declarations as a type qualifier for pointers. It doesn't add any new functionality. It is only a way for programmer to inform about an optimizations that compiler can make. When we use restrict with a pointer ptr, it tells the compiler that ptr is the only way to access the object pointed by it and compiler doesn't need to add any additional checks.

If a programmer uses restrict keyword and violate the above condition, result is undefined behavior. restrict is not supported by C++. It is a C only keyword.

```
// C program to use restrict keyword.

#include <stdio.h>

 // Note that the purpose of restrict is to show only syntax.

//It doesn't change anything in output (or logic). It is just a way for

// programmer to tell compiler about an  optimization

void use(int* a, int* b, int* restrict c)

{

   *a += *c;

    // Since c is restrict, compiler will not reload value at address c in

   // its assembly code. Therefore generated  assembly code is optimized

   *b += *c;

}

 int main(void)

{

   int a = 50, b = 60, c = 70;

   use(&a, &b, &c);

   printf("%d %d %d", a, b, c);

   return 0;       }
```

The mutable storage class specifier in C++ (or use of mutable keyword in C++)

auto, register, static and extern are the storage class specifiers in C. typedef is also considered as a storage class specifier in C. C++ also supports all these storage class specifiers. In addition to this C++, adds one important storage class specifier whose name is mutable.

What is the need of mutable?

Sometimes there is requirement to modify one or more data members of class / struct through const function even though you don't want the function to update other members of class / struct. This task can be easily performed by using mutable keyword. Consider this example where use of mutable can be useful. Suppose you go to hotel and you give the order to waiter to bring some food dish. After giving order, you suddenly decide to change the order of food.

Assume that hotel provides facility to change the ordered food and again take the order of new food within 10 minutes after giving the 1st order. After 10 minutes order can't be cancelled and old order can't be replaced by new order.

See the following code for details:-

```
#include <iostream>

#include <string.h>

using std::cout;

using std::endl;

 class Customer

{

   char name[25];

   mutable char placedorder[50];

   int tableno;

   mutable int bill;

    public:

   Customer(char* s, char* m, int a, int p)

   {

     strcpy(name, s);
```

```
      strcpy(placedorder, m);
      tableno = a;
      bill = p;
     }
    void changePlacedOrder(char* p) const
    {
      strcpy(placedorder, p);
    }
    void changeBill(int s) const
    {
      bill = s;
    }
    void display() const      {
      cout << "Customer name is: " << name << endl;
      cout << "Food ordered by customer is: " << placedorder << endl;
      cout << "table no is: " << tableno << endl;
      cout << "Total payable amount: " << bill << endl;
    }      };
 int main()
{
    const Customer c1("Pravasi Meet", "Ice Cream", 3, 100);
    c1.display();
    c1.changePlacedOrder("GulabJammuns");
    c1.changeBill(150);
 c1.display();
    return 0;    }
```

Enumeration (or enum) in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants,the names make a program easy to read and maintain.

```
enum State {Working = 1, Failed = 0};
```

The keyword ‘enum’ is used to declare new enumeration types in C and C++. Following is an example of enum declaration.

// The name of enumeration is "flag" and the constant are the values of the flag. By default, the values of the constants are as follows:

// constant1 = 0, constant2 = 1, constant3 = 2 and so on.

```
enum flag{constant1, constant2, constant3, ....... };
```

Variables of type enum can also be defined. They can be defined in two ways:

// In both of the below cases, "day" is defined as the variable of type week.

```
enum week{Mon, Tue, Wed};

enum week day;

// Or

enum week{Mon, Tue, Wed}day;
```

// An example program to demonstrate working of enum in C

```
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()

{

   enum week day;

   day = Thur;

   printf("%d",day);

   return 0;

 }
```

Understanding “volatile” qualifier in C | Set 2 (Examples)

The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler.

Objects declared as volatile are omitted from optimization because their values can be changed by code outside the scope of current code at any time. The system always reads the current value of a volatile object from the memory location rather than keeping its value in temporary register at the point it is requested, even if a previous instruction asked for a value from the same object. So the simple question is, how can value of a variable change in such a way that compiler cannot predict.

Consider the following cases for answer to this question.

```
#include<stdio.h>

 enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

 int main()

{

   enum week day;

   day = Wed;

   printf("%d",day);

   return 0;

}
```

// C++ program to illustrate use of “new” keyword

```
#include<iostream>

using namespace std;

class car

{

   string name;

   int num;

    public:

       car(string a, int n)     {

          cout << "Constructor called" << endl;

          this ->name = a;

          this ->num = n;

       }

        void enter()     {

          cin>>name;

          cin>>num;      }

        void display()    {

          cout << "Name: " << name << endl;

          cout << "Num: " << num << endl;

       }   };

    int main()       {

   // Using new keyword

   car *p = new car("Honda", 2017);

   p->display();

}
```

- - - X X X X X X - - -