1 Month
(2+ duration)

Mon-Fri: 7 pm to 9pm
Weekends - Holiday

Week1 - C and C++ (7-8 session)
Week2-4 - IOT Project (from 9 session)

LMS
-login id and password

class Recordings
study materials (ppt , assignments programs)

login issues

support@emertxe.com

Announcement
- To be used by Mentors for sharing informations

Discussion Forum
- ask general /subject related quieries

After Internship

Complete IOT Project
Submit on LMS
Evaluated by Mentors ~ 2 weeks
Receive Certificate

https://moodle.emertxe-group.com/

C programming

- Introduction to C
- Basic Datatypes
- If statements and Loops
- Arrays
- Pointers
-Functions
-strings and storage class
-Preprocessor directive - #include , #define

Introduction to C

High Level - Java , c++ python, scripting - Portable

Middle level - C

Low level - Assembly languange — PIC-STM — MOV LOAD ADD

Hardware — Fast and Efficient

Portablity and Efficiency

```
if  (num==5){

}
```

```
if num==5
then
        i=9
```

comment - non executable statements

```
//    - single line comment
/*
  Name:
  Date:
  Description:
  Sample Input:
  Sample Output:
*/ -  Multiline comment
```

Number System

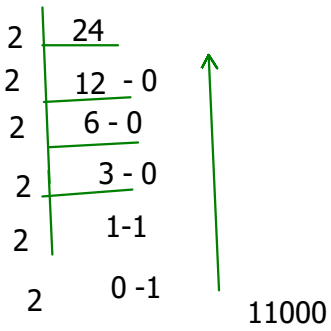
Integer

1.Decimal - Base 10 (0 - 9) 10
2.Binary - Base 2 (0,1) 0b10
3.Octal - Base 8(0-7) 010
4.Hexademical- Base 16 (0-9 ,A-F) 0x10

1 Byte memory - 8 bits
1 bit - binary digits (0/1)

0000 0000 - 1111 1111

Decimal to Binary



Binary to Decimal

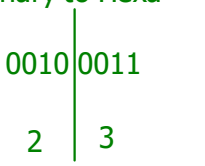
$1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0$

Hexa to Binary

0X45 ---- 0100 0101

-4 bits

Binary to Hexa



0x23

Bit - 0/1
Byte - 8 bits
Word - multiples of bytes
4 bytes - 32 bit
8 bytes - 64 bit

12 -12

0000 1100 2's complement

Steps to find 2's complement

1. Represent in binary
2. take 1s complement
   (convert 1 -> 0 , 0->1)
2.add +1

0000 1100
11
1111 0011
+1

1111 010 0

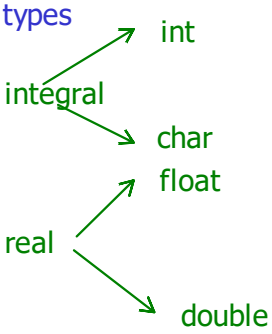
-k = 2^n-k

n= no of bits

Data Representation

int - > +ve - binary format
-ve - 2s complement

Data types



gcc - compiler
app to convert source to machine understandble code

Variable declaration

datatype varaible_name;

Ex

int num; (4 bytes - gcc)
char ch; (1 byte)
float fnum; (4 bytes)
double dnum; (8 bytes)

Modifiers
-size
short (int)
long (int)
based on compiler

short int num ; //2 bytes in gcc

long int num ; // 8 bytes in gcc

-sign
signed (char/int)
unsigned (int/char)

signed int num ; (+ve and -ve)
unsigned int num; (+ve)

int num; (gv)

int num = 10 ;

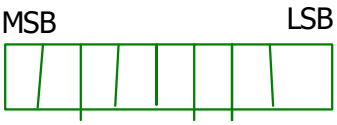
char ch = 'a' ;

float fnum = 45.6;
double dnum = 67.6;

sign modifiers

signed and unsigned

signed char num ;



MSB - sign bit

+ve = 0
-ve = 1

0 000 0000 - 0
0 000 0001
0 000 0010
....

0 111 1111 - 127

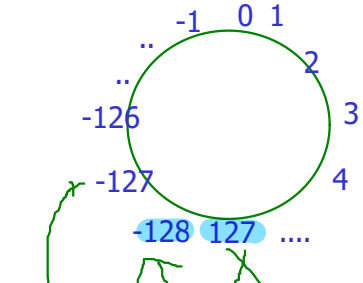
1 000 0000 -128
1 000 0001
1 000 0010
....
1 111 1111 -1

1111 1111
1s c 0000 0000
+1
0000 0001

Range of positive value = 0 to 127

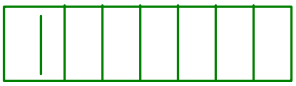
Range of negative value= -128 to -1

signed char -128 to 127



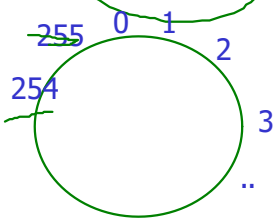
128 -> -128
129 -> -127
overflow

unsigned char num;



0000 0000 0

1111 1111 255



-1 => underflow

overflow
256
257

Range of signed type = - 2^(n-1) to +2^(n-1)-1

Range of unsigned type = 0 to +2^(n-1)

n= no of bits
char , n = 8 bits
int , n = 32 bits

Relational operator

==
!=
<
>
<=
>=

to print output

```
printf("Hello world");
int num=20;

printf("Num value is %d" , num);
```

int - %d
char - %c
float - %f
double - %lf

to read input

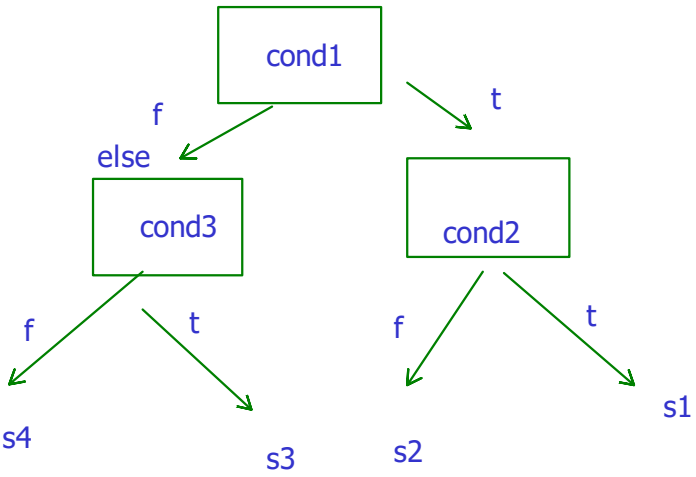
```
int num;
scanf("%d",&num);

char ch;
scanf("%c",&ch);
```

wap to check whether number is positive or negative

```
if (cond )
{
  statemetns;

}
```

Nested if - if inside another if

```
if ( cond1)
{
   if( cond2)
   {
    s1
  }
   else
   {
   s2;
   }
}
else
{
  if( cond3)
   {
    s3
  }
   else
   {
   s4;
   }
}
```



switch case:
-alternative for if else if
- one of many

```
switch(expr/var)
{
   case 1:
    s1
    break;
   case 2:
    s2;
   break;
  case 3:
   s3 ;
  break;
  default:
   s4;
   break;
}
```

case label

case label can be integer value only

wap to implement basic calculator

1.Addition
2.Subraction
3.Multiplication
4.Division

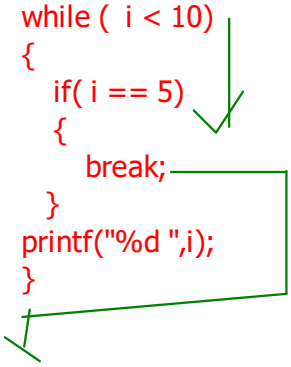
Jump statements

1. break - used inside sw case and loops
2. continue - only inside loops

break - brings control out of loops and swtich case

continue- skip only particular iteration of loop

```
int i = 0;
while ( i < 10)
{
   if( i == 5)
   {
      break;
   }
printf("%d ",i);
}
```



Logical operator

logical AND	&&
logical OR	||
logical NOT	!

cond1 && cond2

cond1	ond2	&&
F	F	F
F	T	F
T	F	F
T	T	T

Logical OR ||

Cond1	cond2	||
F	F	F
F	T	T
T	F	T
T	T	T

Logical Not

cond	!
T	F
F	T

```
num1 = 1;
if (++num1 || num2++)
```

short circuit evaluation

2 ||
T || = T

if (0) - > false
if(non-zero) - true

In logical OR if 1st cond is true , irrespective to the 2nd cond , result of logical OR is true, so 2nd cond is not evaluated , this is short circuit evaluation

In logical AND if 1st cond is FALSE , irrespective to the 2nd cond , result of logical AND is false, so 2nd cond is not evaluated , this is short circuit evaluation

=
+=
-=
*=
/=

num += 1
num = num +1

num *=2;
num = num * 2

```
int num1 = 1, num2 = 1;
float num3 = 1.7, num4 = 1.5;
num1 += num2 += num3 += num4;
printf("num1 is %d\n", num1); //5
```

num1 += num2 += num3 += num4;

num1 += num2 +=(num3 = num3+ num4) //3.2

num1 += (num2 +=num3)
num1 += (num2 = num2 + num3) //1+3.2

num1 += num2;
num1 = num1+ num2// 1+4

num1

Bitwise operators

Bitwise AND &
Bitwise OR |
Bitwise EXor ^
Bitwise complement ~
Bitwise left shit
Bitwise right shift

Bitwise AND

I1	I2	OP
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR

I1	I2	OP
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR

I1	I2	O/P
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Complement

1 0
0 1

Bitwise left shift

num << no of shifts

Bitwise Right shift

num >> no of shifts

num1= 0x45, num2 = 0x51

num1 & num2

```
num1  = 0100  0101
num2  = 0101  0001
--------------------
        0100  0001 =0x41
```

num1 | num2

```
num1  = 0100  0101
num2  = 0101  0001
--------------------
        0101  0101
```

0x55

num1 ^ num2

```
num1  = 0100  0101
num2  = 0101  0001
--------------------
        0001  0100
```

Bitwise complement

num = 0x08 msb is lost, lsb is filled with 0

lsb is lost , msb is filled with 0

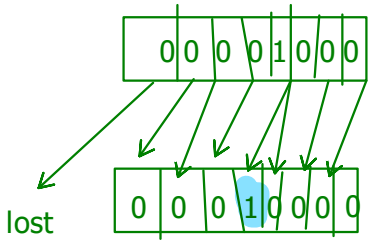
num1=0x55

0101 0101

~num1

1010 1010

num << 2



num<<1

0010 0000 32

2^7 + 2^6 + 2^5 + 2^4 +2^3 + 2^2 +2^1 +2^0

7 6 5 4 3 2 1 0

left shift is an efficient way of multiplying number by 2 power value

num >> 2

0001 0000 = 16

0000 1000 = 8

0000 0100 = 4

divide by 2

sizeof()

float num;

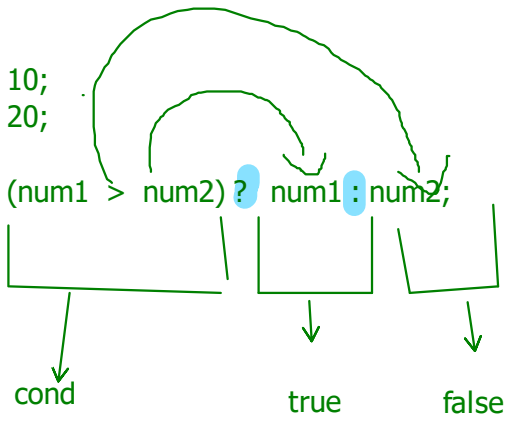
sizeof(int)

sizeof(float)

sizeof(num) ; 4 bytes

?:

```
num1 = 10;
num2 = 20;

num3 = (num1 > num2) ? num1 : num2;
```



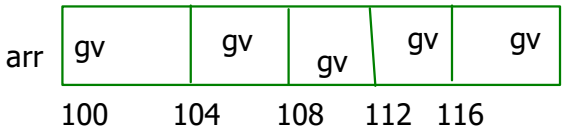
Arrays

collection of elements of similar type
continuous memory allocation

int num1=10;
int num2,..10;

syntax

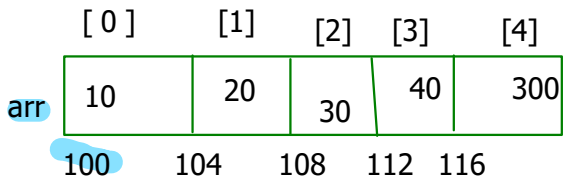
datatype arr_name[size];



int arr[5];

total memory = 5 * 4 = 20

int arr[5] = {10,20,30,40,50};



float a[4];

4*4 = 16

char a[10]; //10 bytes

indexing starts from 0 to size-1

printf("%d" , arr[0]); 10

arr[4]= 300;

printf("%d", arr[2]);

array name indicated base address

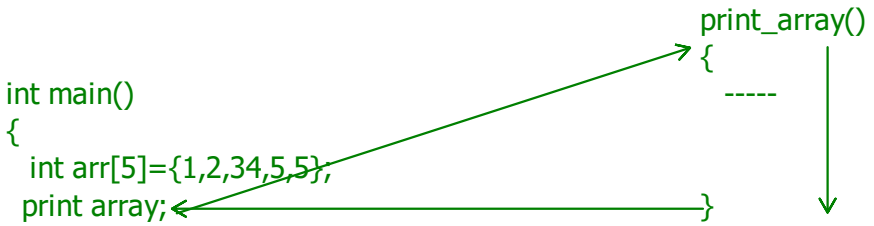
int arr1[5];
int arr2[5];

arr1 = arr2 ; // modifiying base address which is constant

//error

Functions

Set of Statements for specific task



1.resuse code
2. optimized code
3.Easy to understand
4. Easy to test and debug
5.Modularity

1.Function definition
2.Function call
3.Function declaration

How to write a function

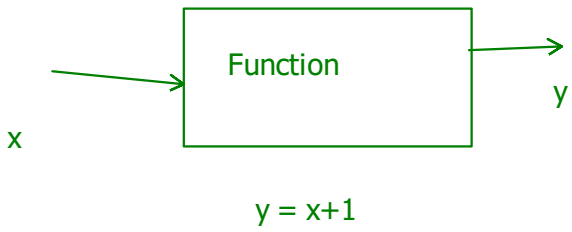
```
return datatype  function_name( datatype arg1 , datatype arg2 ,.. datatype arg n)
{
  /*function body*/
}
```

How to call function

```
function_name(arg1,arg2,..argn);
```

How to declare function (Function declaration , function prototype)

```
return datatype  function_name( datatype arg1 , datatype arg2 ,.. datatype argn);
```



y = x+1

```
int foo(int x);

int foo(int x)
{
   int y;
   y = x+1;
   return  y;
}
```

```
int main()
{
    int res;
  res =foo(5);
  printf(%d\n",res);
}
```

wap to add 2 numbers using function

i/p : 2 3
o/p : 5

Pass by Value
Pass by reference

1.Pass by value
value is passed ,
function value in received in another variable(value is copied)

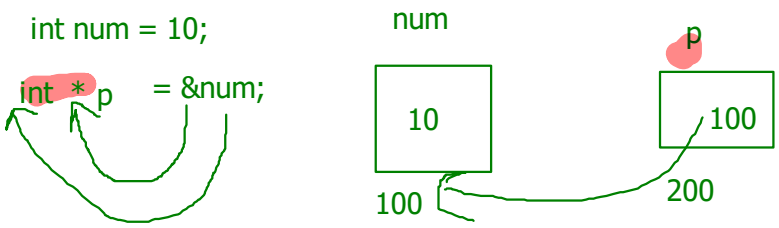
- to reflect the changes , vaule shoulb be returned
- return only one value from the function

2. Pass by reference

-address of variable is passed ;
-changes is done on formal parameter
- return multiple values from function

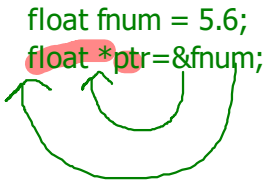
Pointers

pointer is type to store address of other variable

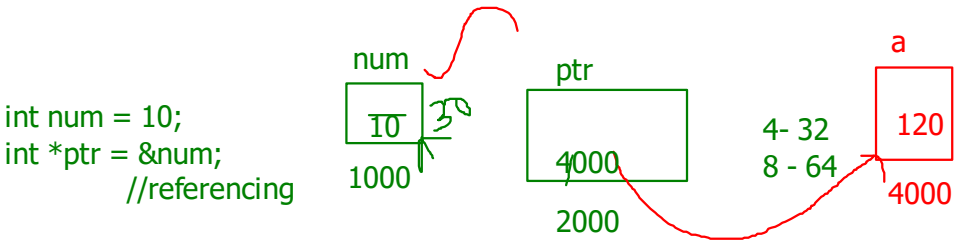


size of address depends on bitness of the system
32 bit - 4 bytes
64 bit - 8 bytes

p is a pointer pointing to address of int variable



cptr is pointing to char variable



printf("%d",*ptr);
//10 dereferencing

*ptr = 30;

ptr = &a

*ptr

%p - print address (hexa format)

printf and scanf
- portable

pointers -> middle level to low level - efficiency
functions -> middle level to high level - portable

```
int num = 10;
char *p  =&num;
```

```
char *ptr1;     //4  - fetch 1 byte
int *ptr2;      //4  - fetch 4 bytes from mem
```

why do we need differnt datatype for pointer variables?

```
char *p  =&num;  // warning
```

*p => fetches only one byte from mem

```
int num = 0x12345678;
```

Little Endian -lsb bytes is stored in lowest address

Endians

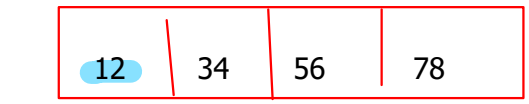
how data is stored in memory

byte ordering

-depends on hw

Big endian - msb byte is stored in lowest address

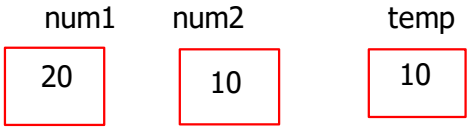
```
int num = 0x12345678;
```



wap to swap 2 numbers using function

i/p : num1 = 10
num2 = 20

o/p : num1 = 20
num2 = 10;




```
temp = num1;
num1 = num2;
num2 = temp;
```

int num=10;
&num -> to get address

int arr[5] ={1,2,3,4,5};

arr - base address

int *ptr

int *ptr ; //gv
//wild pointer
printf("%d",*ptr);

4/8 bytes

ptr 12345

12345
value

int num = 0 ; //gv

cant predict
illegal access - segmentation fault

undefined behaviour

wild pointer
An uninitialized pointer pointing to random memory is called as wild pointer

int *ptr = 0; //address 0

int *ptr = NULL;

address 0 - reserved for os

printf("%d",*ptr); -> segmentation fault
predictable behaviour

pointer initialized with address 0 is called as NULL pointer

NULL Pointer points nothing(not pointing to any valid address)

```
if( ptr != NULL)
    printf("%d",*ptr);
```

```
const int num =10;
num = 30; //error

const int * ptr = &num;

    ptr  = address - varying
    *ptr =  value - const

*ptr = 45; //error
ptr = &num2;
```

Pointer to const

```
int *const ptr =&num;

address - const
value   -varying

*ptr = 30;   valid

    ptr = &num3;//error
```

const pointer

segmentation fault - dereferencing NULL pointer / access memory which is not permite
(run time - at the time of execution)

undefined behaviour - gv / run time error

if you try to access illegal memory which is not allocated for the program

strings

collection of characters terminated by null character '\0'

continuous memory

char ch='a';

char vowels [] ={'a','e','i','0','u'}; //collection of chars
not a string. array of char

char vowels[]={'a','e','i','0','u','\0'}; //string

char str[]="hello"; //by default null char is added

string

"hello"

printf("Hello world"); 'a' - 97
'A' - 65

printf("hello" "world"); - > "helloworld"

char str[]="Hello" ;	char *s= "Hello";
	read only segment
str H e l l o \0	s 200 → 200 Hello
100	
sizeof(str) = > 6 bytes	sizeof(s) => 4/8
modifiable string str[0] ='h';	non modifable string or string literal s[0]='a' - error

storage classes

```
int x ; //global variable
int main()
{
  int num1;      //local variable

}
int fun(int a, int b)  //function parameter
{
 int sum; //local variable
}
```

local variables are declared inside function block
global variables are declared outside the block

- to tune the propery of variable using storage class
  - memory location
  - scope - where we access ?
  - life time - when it is created and

    when it is deleted?

memory segements

RAM

-stack - local variables . function parameters
-heap - dynamic memory allocation
-data segment - global variables and static
-code or text segment - exe file , string literals (read only)

int x =10; // initialized DS
int x; // Block Started by Symbol (uninitialized DS)

x =0 (by default)

		memory	scope	lifetime
local	auto	stack	block	block entry &exit
	register	Register	block	block entry &exit
	static	Data	Block	from start till the end of the program .entire program
global	static	Data	file	from start till the end of the program
	extern	Data	program	from start till the end of the program

```
int =10;// global
int fun();
 int main()
 {
  static int num; // 0  BSS

 }

  int fun()
  {
   num= num+2;

  }
```

static local is used to retain the value of variable over multiple function calls

static global can be accessed only inside a file
extern global can be accessed from other file

```
while(1)
if(non- zeor)-true
if(0)- false
```

stdio.h
string.h

prog.c
add.c

#

vs

funtions declaration

global variable declaration

UDT
MACRO Definiton

reusable function /var declaration

funciton and variabledefinitions

non sharable
non reusable code

#include
preprocessor directive

< > -std file

" " -used defined header file

#ifndef MACRO
#define MACRO
<declaration>

#endif

if header file is already included in program or not , if included , then dont include
if it is not included , then the code is included in program

used to include header file only once in program

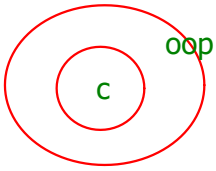
macro is name for constant value

line #define <macro_name> <const>

function like macro
multiline macro

SWAP(num1,num2)

c++



c++ is super set of c

C - POP

C++ - OOP

```
main()
{

}
```

design
main()

Object

Real world entity

Laptop, mobile , car , pen ,
Dog, cat, human being

Every object has State and Behaviour

state - property
car - wheels , brand name, engine, colour , dimensions

dog - bread , height, colour , weight, age

Behaviour -actions

start , drive , stop

dog - bark, jump, run , bite

Every object has an identity

Mobile banking - object

state

acc type
acc name
Balance
address
phone no

id

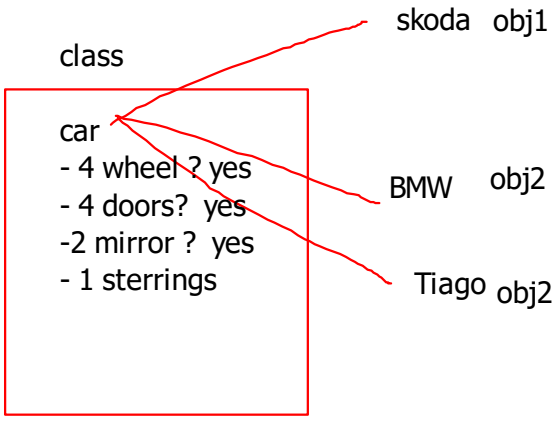
acc number

Behaviour

tranfer
check_baalnce
apply_loan

class is a used defined data type
Objects are created from class

class - blueprint



1 class - n object

syntax

```
class  class_name
{
   //state
   - data members
   //behaviour
    //member functions


};
```

class_name obj_name;

size of object depends on data members

}

Access specifiers

1.private - (by default) data members and member functions can be accessed only within the class
2.public - can be accessed from anywhere (any fucntion , outside class)
3.protected - same as private , cannot be accessed outside the class, can be accessed onl inside class and from derived class

constructor function
used to initialize object
- special member function (part of class)
- automatically called when obejct is created
- must be defined under public
- by default compiler adds constructor function with empty body

```
int main()
{

 func();
}

void func()
{
   student s1;
}
```

Destructor
- delete dynamic memory allocation .
special member function (part of class)
- automatically called when object comes out of the scope
-must be defined under public by default compiler adds destructor function with empty bodyd

Parameterized construtor

```
class  Student
{
 public:
  int id;
 int marks;
 Student()
{
   id =100;
}
 int display()
int main(){
{
  student obj1 ;
 }
 };
   obj1.id  =100;   //private -inaccessible
    obj1 . display();
}
```

Pillars of OOPS

1. Encapsulation
   Binding data members and member function together into single entity is called encapsulation.

```
class  test
{
  int id;
 int display();

};
```

2.Abstration - data hiding

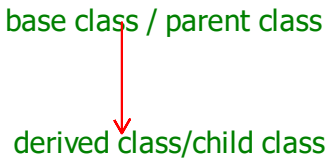
private access specifiers

Inheritance

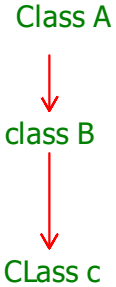
creating new class with inheriting the properties from already existing class

- creating child class from parent class

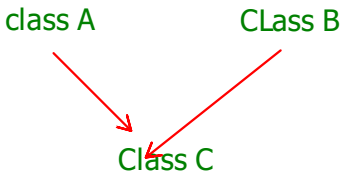
Single Inheritance



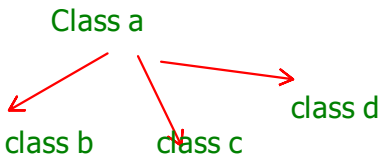
Multilevel Inheritance



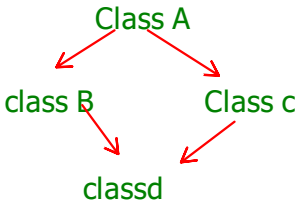
Multiple Inheritance



Hierarchial Inheritance



Hybrid



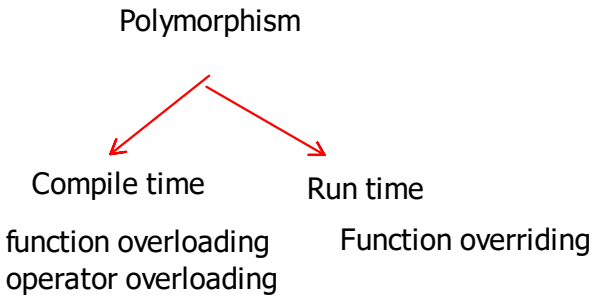
syntax

```
class  parent_class_name
{

};

class child_class_name : public  parent_class_name

{


};
```

```
class child_class_name : public  parent_class_name
,public parent2_class nmae
{


};
```

Polymorphism
-many forms

-data /member functions




```
add(3,4);                    int add( int  num1 , int num2 )
                             {
add(5.4,6.7)                    return num1+num2;
                             }

  add( 3,4,5)
                             float add(float num1,  float num2)
                             {

                             }
```

same function nmae
differnt number of args
or differnent
arg datatype

```
                             add(int num1, int num2, int num3)
                              {

                              }
```

Function overriding

refefining same function in child class which is alreay part of parent class.