1.

oops concepts:

a.encapsulation

b.data abstraction

c.inheritance

d.polymorphism

Encapsulation:

Data abstraction;

it means data hiding i.e making data protected,public and private.

Inheritance:

it means inheriting the properties of one class to another class

polymorphism:

it means having many forms,i.e calling one function will execute some other function.

It is two type:

compile time and run time.

->function overloading,operator overloading will comes under compile time polymorphism

->virtual functions will comes under run time polymorphism.

2.

Copy constructor:

if we want to copy one object data to another object we uses copy construct

#include<iostream>

using namespace std;

class deep

{

int v1,v2;

public:

deep()

{

v1=12;v2=10;

}

deep(deep &s)

{

v1=s.v1;

v2=s.v2;

}

void setdata(int a,int b)

{

v1=a;v2=b;

}

void getdata()

{

cout<<"v1:"<<v1<<endl;

cout<<"v2:"<<v2<<endl;

}

};

int main()

{

deep obj1;

cout<<"obj1 output"<<endl;

obj1.getdata();

deep obj2(obj1);

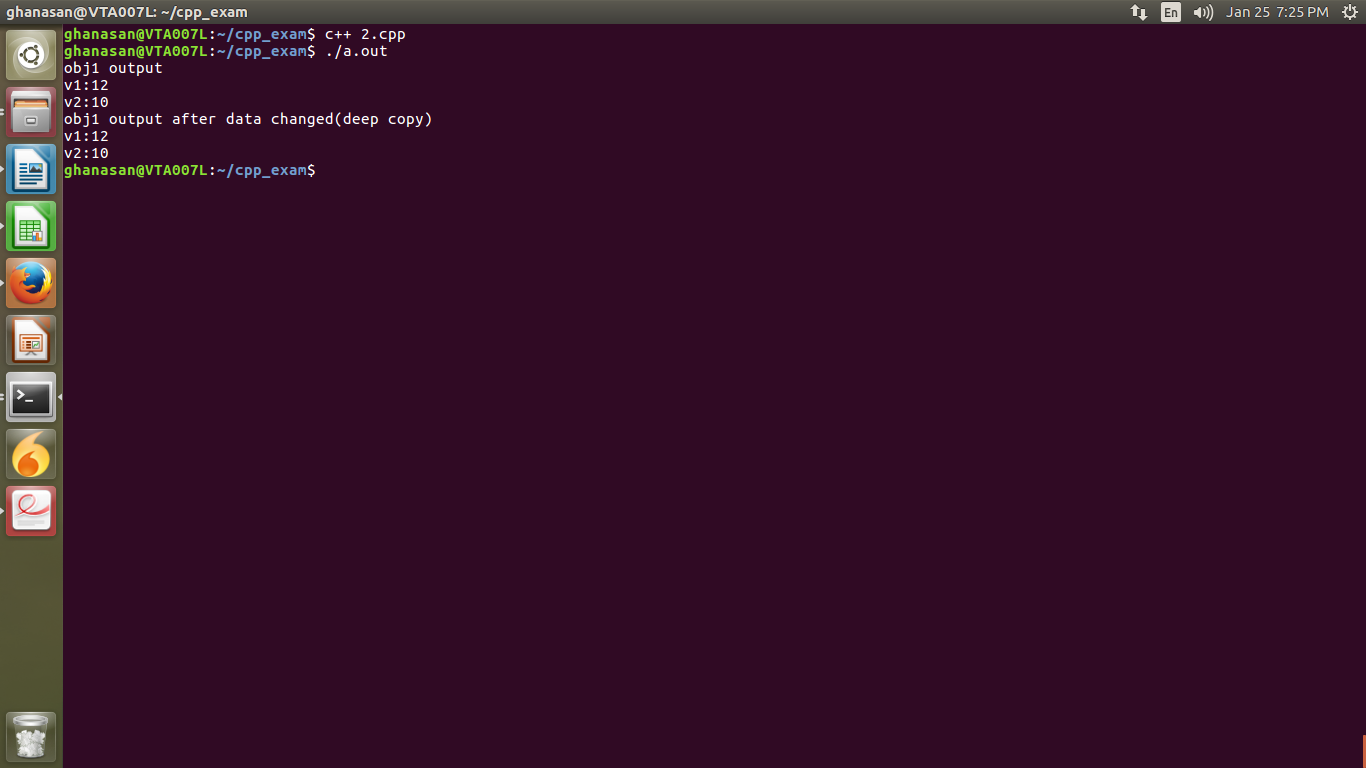
// obj2=obj1;

obj2.setdata(1,2);

cout<<"obj1 output after data changed(deep copy)"<<endl;

obj1.getdata();

}



3.

Inheritance:

inheriting the base class properties into derived class i.e reusing the code of base class in derived class instead of rewriting the code.

Multiple ineritance:

inheriting from more than one class.

Virtual function:

it is the function which is diclared as virtual,calling one function will execute other function

#include<iostream>

using namespace std;

class base

{

public:

virtual void fun()

{

cout<<"base class"<<endl;

}

};

class derived:public base

{

public:

void fun()

{

cout<<"derived class"<<endl;

}

};

int main()

{

derived \*ptr;

ptr=new base [];

ptr->fun();

}

4.

STL:

standerd template library has algorithms,iterators,containers.these are used for generic perpose

->templates are two type,function templates and class templates

5.

#include<iostream>

using namespace std;

class Time

{

int h,m,s;

public:

Time(int v1=0,int v2=0,int v3=0)

{

h=v1;m=v2;s=v3;

}

void settime(int v1,int v2,int v3)

{h=v1;m=v2;s=v3;}

void print()

{

if(h>23||m>59||s>59)

{

throw "Exception:invalid invalid time";

}

cout<<h<<":"<<m<<":"<<s<<endl;

}

void sethour(int v)

{

h=v;

}

void setminute(int v)

{

m=v;

}

void setsecond(int v)

{

s=v;

}

int gethour()

{return h;}

int getminute()

{return m;}

int getsecond()

{return s;}

void nextsecond()

{

if(s+1>59)

{

s=0;

m++;

if(m==60)

{

m=0;

h++;

if(h==24)

h=0;

}

}

else

s=s+1;

}

};

int main()

{

Time t1(23,59,59);

t1.print();

t1.sethour(12);

t1.setminute(30);

t1.setsecond(15);

t1.print();

cout<<"hour is:"<<t1.gethour()<<endl;

cout<<"minute is:"<<t1.getminute()<<endl;

cout<<"second is:"<<t1.getsecond()<<endl;

Time t2;

t2.print();

t2.settime(1,2,3);

t2.print();

Time t3(12);

t3.print();

Time t4(23,59,58);

t4.print();

t4.nextsecond();

t4.print();

t4.nextsecond();

t4.print();

Time t5(25,61,99);

try{

t5.print();

}

catch(const char \*error)

{

cout<<error<<endl;

}

}

