Assignment:

*OOPD:*

**By Surbhi , MT19055**

1. a.

#include <stdio.h>

#include <stdlib.h>

#define pi 3.14

float area(float r,float l)

{

float a;

a=2\*pi\*r\*(r+l);

return a;

}

float volume(float r,float l)

{

float v;

v=pi\*r\*r\*l;

return v;

}

float pcost(float a,float c)

{

float pc;

pc=a\*c;

return pc;

}

int main()

{

float rad,len,ar,vol,cost,c;

printf("Enter radius of cylinder: ");

scanf("%f",&rad);

printf("Enter length of cylinder: ");

scanf("%f",&len);

ar=area(rad,len);

vol=volume(rad,len);

printf("Enter cost of painting per sqmeter: ");

scanf("%f",&c);

cost=pcost(ar,c);

printf("Surface area of cylinder: %.3f",ar);

printf("\nVolume of cylinder: %.3f",vol);

printf("\nCost of painting the cylinder: %.3f",cost);

return 0;

}

1) b.

#include <stdio.h>

#include <stdlib.h>

#define pi 3.14

struct cyl{

private:

float radius;

float height;

public:

float ar,vol,cost,pc;

void area();

void volume();

void cal\_pcost();

void set\_radius(float r){

radius=r;

}

void set\_height(float h){

height=h;

}

float get\_radius()

{

return radius;

}

float get\_height()

{

return height;

}

};

cyl c;

void area()

{

c.ar=2\*pi\*c.get\_radius()\*(c.get\_radius()+c.get\_height());

}

void volume()

{

c.vol=pi\*c.get\_radius()\*c.get\_radius()\*c.get\_height();

}

void cal\_pcost()

{

c.cost=c.ar\*c.pc;

}

int main()

{

float rad,ht;

printf("Enter radius of the cylinder: ");

scanf("%f",&rad);

printf("Enter height of the cylinder: ");

scanf("%f",&ht);

c.set\_radius(rad);

c.set\_height(ht);

area();

volume();

printf("Enter cost of painting cylinder per sqmeter: ");

scanf("%f",&c.pc);

cal\_pcost();

printf("Surface\_Area of Cylinder: %.3f\n",c.ar);

printf("Volume of Cylinder: %.3f\n",c.vol);

printf("Cost of painting the cylinder: %.3f",c.cost);

return 0;

}

1) c.

Program ‘b’ is better than Program ‘a’ because in this data and operations on that data are wrapped into one single unit and only the interface of the operation is visible i.e. surface area, volume, cost of painting. In this changes in implementation does not affect objects i.e. calculation of surface area, volume and cost of painting works even if radius and height components are changed.

2) a.

#include <stdio.h>

#include <stdlib.h>

struct student

{

int sid;

int marks[6];

};

float cal\_avg(int m[])

{

int i,sum=0;

float a;

for(i=0;i<6;i++)

{

sum=sum+m[i];

}

a=sum/6;

return a;

}

int cal\_best(int m[])

{

int i,max;

max=m[0];

for(i=0;i<6;i++)

{

if(m[i]>max)

{

max=m[i];

}

}

return max;

}

int cal\_worst(int m[])

{

int i,min;

min=m[0];

for(i=0;i<6;i++)

{

if(m[i]<min)

{

min=m[i];

}

}

return min;

}

int main()

{

struct student s1;

int i,best,worst;

float avg;

printf(“Enter student id: “);

scanf(“%d”,&s1.sid);

for(i=0;i<6;i++)

{

printf("Enter marks in subject %d : ",i+1);

scanf("%d",&s1.marks[i]);

}

avg=cal\_avg(s1.marks);

best=cal\_best(s1.marks);

worst=cal\_worst(s1.marks);

printf("\nAverage marks: %.2f",avg);

printf("\nBest marks: %d",best);

printf("\nWorst marks: %d",worst);

return 0;

}

1. b.

#include <stdio.h>

#include <stdlib.h>

struct student

{

private:

int sid;

int marks[6];

public:

float avg;

int best,worst;

void cal\_avg();

void cal\_best();

void cal\_worst();

void set\_sid(int i)

{

sid=i;

}

void set\_marks(int m[])

{

int i;

for(i=0;i<6;i++)

{

marks[i]=m[i];

}

}

int get\_id()

{

return sid;

}

int\* get\_marks()

{

return marks;

}

};

struct student s1;

void cal\_avg()

{

int i,sum=0;

for(i=0;i<6;i++)

{

sum=sum+s1.get\_marks()[i];

}

s1.avg=sum/6;

}

void cal\_best()

{

int i,max;

max=s1.get\_marks()[0];

for(i=0;i<6;i++)

{

if(s1.get\_marks()[i]>max)

{

max=s1.get\_marks()[i];

}

s1.best=max;

}

}

void cal\_worst()

{

int i,min;

min=s1.get\_marks()[0];

for(i=0;i<6;i++)

{

if(s1.get\_marks()[i]<min)

{

min=s1.get\_marks()[i];

}

s1.worst=min;

}

}

int main()

{

int i,id,mrk[6];

printf("Enter student id: ");

scanf("%d",&id);

for(i=0;i<6;i++)

{

printf("Enter marks in subject %d : ",i+1);

scanf("%d",&mrk[i]);

}

s1.set\_sid(id);

s1.set\_marks(mrk);

cal\_avg();

cal\_best();

cal\_worst();

printf("\nAverage marks: %.3f",s1.avg);

printf("\nBest marks: %d",s1.best);

printf("\nWorst marks: %d",s1.worst);

return 0;

}

1. c.

Program ‘b’ is better than Program ‘a’ because in this data and operations on that data are wrapped into one single unit and only the interface of the operation is visible i.e. surface area, volume, cost of painting. In this changes in implementation does not affect objects i.e. calculation of surface area, volume and cost of painting works even if radius and height components are changed.

3)

State of the object

Change Number

Change Street Name

Change City

Change State

Passport Number = 27

Name = Raman Venkateshan

Date of Birth = 15 July 1987

Address :

Number = 47

Street = Bhaskar Street

City = Malgudi

State = Tamil Nadu

Interface

Op1

Op2

Op3

Op4

Alternatively, an object is a collection of operation that share a state.

Change Number

Change Street

Change City

Change State

**Passport Number = 27**

**Name = Raman Venkateshan**

**Date of Birth = 15 July 1987**

**Address : Number = 47**

**Street = Bhaskar Street**

**City = Malgudi**

**State = Tamil Nadu**

4)a.

Before Call :

Main function Local variables X,Y

After call :

X,Y

X

Return Address

Local variables of Temp

Top of Stack

Temp(X)

MAIN

SECOND

**b).**Message has 4 parts as following:

1. Identity of the recipient object-Second
2. Code to be executed by the recipient or the method-Temp(x)
3. Arguments for the code-X
4. Return value- no return value.

c).

Message passing is preferred over function calling because:

**Message passing** can also refer to the way one calls methods in languages like objective C. Languages like C think of it as message passing, but it's a subroutine that involves pushing something on the stack while suspending the method calling the subroutine. The implementation is stack passed method invocation which implies the caller is blocked while it waits on the calling routine to return. Message passing has no such semantics. It doesn't have to wait for the routine its invoking to finish.

So advantage of message passing over function is that in message passing the structure of called object need not be known and message can simply be send while in case of function calling some reference to the called object should be know beforehand.