Mid-Sem - 2016

Q(a) ** Need for OOP // Features of OOP.

1) It reduces complexity
Code organised into classes, which is easier.

2) It provides reusability
Code can be reused thru inheritence of classes.
Derived class can inherit ~~f~~ code of base class.

3) It provides data hiding
Variables are hidden inside a class by using private. It helps to avoid unintentional modification of data.

3) It provides poly-morphism
PM means using same entity in multiple ways. Same fun & operators can be used in multiple ways.
It is done thru fun & operator overloading.

4) It provides abstraction
Abstraction means to use a class, without knowing the implementation details.
Objects of a class can be directly used to call the fun without knowing how they work.

5) It is based on real world
So obj & classes are based on real world objects. So it is easy to understand

ii) 5 appl of C++ :-
Real time system
Simulation & modeling
Obj oriented database
Hypertext, hypermedia & expertext
AI & expert system.

Q1 b) Ques on electricity board.

```
class el
{       string name;
        int units;
        int charges;
public:
void set(string x, int y)
void calcharges();
void get();
};
void el :: set(string x, int y)
{
        name = x;
        units = y;
}
void el :: calcharges()
{
        if(units < 100)
        {
                charges = .6 * units;
                if(charges < 50)
                {
                        charges = 50;
                }
        }
        if(units > 100 && units < 300)
        {
                charges = 100 * .6 + (units - 100) * .8
        }
```

```
if (units > 300)
{
    charges = 100 * .6 + 200 * .8
              + (units - 300) * .9

    if (charges > 300)
    {
        charges = charges + .15 * charges;
    }
}
void el :: get ()
{
    calcharges ();
    cout << name << units << charges;
}
int main ()
{
    int n;
    cout << " How many users you want ";
    cin >> n;
    el * arr_obj = new el [n];
    cout << " enter details of n obj ";
    string s; // for name
    int i; // for units
    for (i = 0 to n)
    {
        cin >> s >> i
        arr_obj [i] . set (s, i);
    }
    cout << " Print details of n obj ";
    for (i = 0 to n)
    {
        arr_obj [i] . get ();
    }
}
```

b)ii) 5 feat of OOPs -<done>

(iii) Ques on voltage

```
#include <iostream>
#include <cmath> // for sqrt fun.

int power(m, n) // to cal m^n
{
    int result = 1;
    for (int i=0; i<n; i++)
    {
        result = result * m;
    }
    return result;
}

float Cal_V (int n, float f)
{
    float a, denominator, b, V
    a = 23*23 + .25*f*f.
    denominator = sqrt(a);
    b = (275.0) / denominator;
    V = power(b, n);
    return V;
}

int main()
{
    float y;
    int n, f;
    cout << "enter n & f";
    cin >> n >> f;
    y = Cal_V(n,f);
    cout << "at a freq of " << f << " the volt
    gain is " << y;
}
```

Q2 a) Namespaces

- A namespace is a declarative region.
- If you want to have multiple entities with same name like var, fun, etc you can put them under different name spaces
- Thus namespace can avoid name collisions
- To use any entity from a namespace use :: or "using" directive

eg

```
std:: cout << a; // use cout from
                    std ns;
using namespace std;
    // use std namespace for all
    std::cout
    std::cout
```

↑ not required now.

b) Early binding vs late binding
<see folder>

c) Struct vs class
<done in 2018>

d) manipulators - <" " " ">.

e) reference var - <see notes>

Q3 b)

```
class restra
{   int Fcode;
    string food;
    string ftype;
    string sticker;
public:
    void getS();
    void getf();
    void showf();
};
void restra::getf()
{
    cout << "enter values";
    cin >> fcode >> food >> ftype;
    getS();
}
void restra::getS()
{
    if(ftype == "veg")          sticker = "green";
    if(ftype == "contain egg")  "  = "Yellow";
    if("  " == "Non veg")       "  = "Red";
}
void restra::showf()
{
    cout << Fcode << food << ftype << sticker;
}
int main()
{
    restra obj;
    obj.getf();
    obj.showf();
}
```

Q4 (i) i) Dynamic in Static initialization means to provide values at compile time, not during run time.

eg 1) student obj1 (1, 11, 11);

2) student obj1;
obj1.set (1, 11, 11);

Values r provided at compile time

Dy: initialization means to provide values during run time. You need to take values as input from user, then assign to obj.

It can be done by using construction or by set fun.

int a, b, c cin >> a >> b >> c Student obj1 (a, b, c);	int a, b, c; cin >> a >> b >> c; student obj1; obj1.set(a, b, c);
(using const)	(using set fun)

(ii) inline - (See notes)

(iii) const functions

- A class can have const fun.
- These fun r declared like this

```
void get () const
{
    =
    =
}
```

- A const fun can change/modify any var.
- It is only allowed to read/print these var, but not change/assign the var.
- So the answer is const.

Q2 (c) (iv) Yes. Non member fun can access the data member of this class.

This can be done by using a friend fun.

< see notes >

(c) (v) fun(a, b) is call by value

fun(&a, &b) is call by ref.

Q2 (d)

```
class candidate
{
        long rn;
        string cname;
        float age_marks;
        char grade;
        void setG();
        public:
        candidate();
        get_data();
        disp();
};
void candidate::setG()
{
        if(age_marks >=80) grade = 'A';
        if(age_marks < 80 &&
           age_marks >= 65) grade = 'B';
```

```
if (age_marks < 65 &&
    age_marks >= 50
        Grade = 'C';

if (age_marks < 50)
        Grade = 'D';

}

Candidate :: Candidate ()
{
        rn = 0;
        cname = "N.A";
        age_marks = 0.0;
}
Void candidate :: get_data()
{
    long x;
    string y;
    float z;
    cin >> x >> y >> z;
    rn = x;
    cname = y;
    age_marks = z;
    setG ();
}
```

```
void Candidate:: disp( )
{
    cout<<rn<<name<<age<<sex
        <<grade;

}

int main( )
{
    Candidate obj;
    obj.getdata( );
    obj.disp( );

}
```

Q3 a) Types of Constructor
- Parameterized
- Non Parameterized
- Copy Constructor

(Copy Constructor) – (see notes)

Constructor can't be a private bcz it is called when creating obj. If it is private, then it can't be called.

Q36) Class employee

```
{
    string name;
    int id;
    float sal;
  public:
void enter( );
void disp( );
void searchid(int x);
};
void employee :: enter ( )
{
    cout<<"enter details";
    string x;
    int y;
    float z;
    cin>>x>>y>>z;
    name = x;
    id = y;
    sal = z;
}
void employee :: disp ( )
{
    cout<<name<<id<<sal;
}
int employee :: searchid (int x)
{
    if (id == x)
    {
      cout<<"put details";
      disp();
      return 1;
    }
    else return 0;
}
```

```
int main()
    int n, id;
{   cout<<"How many obj";
    cin>>n;
    employee * arr_obj = new employee[n];
    for(i=0 to n)
    {
        arr_obj[i].enter();

    }
    cout<<" enter id to search";
    cin>>id;
    int found=0;
    for(i=0 to n)
    {
        if(arr_obj[i].searchid(id))
        {
            found=1;
            cout<<"Id found"
            break;
        }
    }
    if(found==0)
        cout<<" not found"
}
```

Q3 c). • Overloading is a form of polymorphism.
• overloading means to create multiple use of any entity like fun or operator.
• We can have fun o/L^g & operator o/L^g.

Fun o/L^g – (See notes)

Q3 d)(i) // Find o/p or error.

```
Class Point
{
    float x, y;
};
```

Q3 d). Find o/p or error
• There are some typo. Those r not errors. Correct ques is this

```
Class Point
{
    private:
    float x, y;
    public:
    Point(float Temp X=0,
          float Temp y=0)
    {
        x = Temp X;
        y = Temp y;
    }
    ~Point()
    {
        cout << x << y << "Destroyed";
    }
};
```

int main()
{
(1) Point Point1(25);
(2) Cout << "Good Dues" << endl;
(3) Point Point2(4, 5);
(4) Cout << Sizeof(Point1) <<
Sizeof(Point);

} (5)

// There is no error. Ignore typos.

Line 1: Point1 obj has only 1 param,
so second param will be taken
from default arg wh. is 0.
so Point1 = 25, 0.

line ; 2 :- "Good Dues"

; 3 :- Point2 obj has both param.
So ignore def arguments.
Point2 = 4, 5.

; 4 - Size of obj or class is the
sum size of all variables
There are 2 ints, so size = 8 Bytes.
o/p = 8 8

; 5 :- Destructor called in reverse order
~~o/p~~ Point2 destroyed, its value
is printed in destructor.
∴ So o/p Point2 : 4, 5.
Point1 : 25, 0

O/P will be :-

Good Ques

8 8

4 5 destroyed

25 0 destroyed.

ii)

```
int a = 5;
int main()
{
    int a = 9;
    cout << a; //9
}
    int a = 7, b = 6;
    int c = 4 + 13 ^ 2 - 1 $$ ::a
    cout << a << c; //7, 1
}
cout << a << ::a; //-9, 5
}
```

//No error.

Explanation

c = 4 + 13 ^ 2 - 1 $$ ::a

Order of precedance ::

+, - (+ is on left, so it is evaluated 1st)

^

$$

c = 4 + 13 ^ 2 - 1 && ::a

↓

(4 + 13) ^ 2 - 1 && 5 // Take global a.

↓

17 ^ (2 - 1) && 5

↓

(17 ^ 1) && 5

↓

// ^ is xor operator.

Convert 17, 1 to binary & perform XOR.

17 = 1 0 0 0 1

1 = 0 0 0 0 1

XOR = 1 0 0 0 0

= 16.

↓

16 && 5

↓

Both are non zero, so o/p = 1

∴ (1)

∴ o/p = -9

7 1

-9 5

Early binding refers to events that occur at compile time. In essence, early binding occurs when all information needed to call a function is known at compile time. (Put differently, early binding means that an object and a function call are bound during compilation.) Examples of early binding include normal function calls (including standard library functions), overloaded function calls, and overloaded operators. The main advantage to early binding is efficiency. Because all information necessary to call a function is determined at compile time, these types of function calls are very fast.

The opposite of early binding is late binding. Late binding refers to function calls that are not resolved until run time. Virtual functions are used to achieve late binding. As you know, when access is via a base pointer or reference, the virtual function actually called is determined by the type of object pointed to by the pointer. Because in most cases this cannot be determined at compile time, the object and the function are not linked until run time. The main advantage to late binding is flexibility. Unlike early binding, late binding allows you to create programs that can respond to events occurring while the program executes without having to create a large amount of "contingency code." Keep in mind that because a function call is not resolved until run time, late binding can make for somewhat slower execution times. However today, fast computers have significantly reduced the execution times related to late binding.