Some important extra questions and answers

Composition vs inheritance

>>Composition is considered as simpler in comparision to inheritance. Composition clearly indicates what operations are provided, and what are not.

>> Inheritance provides increased functionality, as many more methods may be inherited from the parent class then are actually being used in the given problem.

>>Increased functionality can also be the disadvantage as these operations are not documented in the code, and so the programmer must examine the parent classes to see what behavior is provided.

>> Composition binds the older abstraction, so it is easier to change the underlying details.

>>Inheritance **permits polymorphism**. Composition does not.

>>Composition makes program longer, and hence **increases complexity**, while inheritance makes for longer and larger hierarchies, and also increases complexity. Which is more problematic differs from situation to situation.

>>Finally, because composition generally introduces an additional function call, there is a small, very small, **execution time advantage for inheritance**.

Forms of Inheritance

1. **Subclassing for specialization**:Here,the new class is a specialized form to the parent class but satisfies the specifications of the parent in all relevant aspects.This form supports the principle of substitutability.Eg:a class **window** provides general windowing operations(like moving,resizing,and so on).A specialized subclass **TextEditWindow** inherits the window operations and in addition provides facilities that allow the window to display textual material and the user to edit text values.
2. **Subclassing for Specification**:Here,the classes maintain a certain interface i.e. they implement the same methods.The parent class can be a combination of implemented operations and operations that are inherited to the child class.Often,there is no any interface change between the parent class and the child class .
3. **Subclassing for Construction**:Here,a class often inherit almost all of its desired functionality from a parent class,perhaps changing only the names of the methods used to interface to the class,or modifying the arguments in a certain fashion.

4. **Subclassing for Combination** :It is sometimes useful to be able to define a new class by inheriting properties from many superclasses; this is known as multiple inheritance . Multiple inheritance brings with it the complexity and with it subclassing for combination can be achieved.
5. **Subclassing for Variance**:It is employed when two or more classes have similar implementations but do not seem to possess any hierarchical relationship between the abstract concepts represented by the classes.Eg:code necessary to control the **mouse** ,may be nearly identical to the code required to control a **graphics tablet**,but there is no reason why class **mouse** should be made the subclass of **graphicstablet**.
6. **Subclassing for Generalization**:Here,a subclass extends the behavior of the parent class to create a more general kind of object. It is applicable in those situations when we build on a base of existing classes that we do not wish to modify or cannot modify.
7. **Subclassing for extension**:Here,the child class must override at least one method from the parent and the functionality is tied to that of the parent.It simply adds new methods to those of the parent ,and the functionality is less strongly tied to the existing methods of parent class.
8. **Subclassing for limitation**:It occurs when the behavior of the subclass is smaller or more restrictive than the behavior of the parent class.It occurs most frequently when a programmer is building on a base of existing classes that should not,or cannot be modified.

Memory Errors

Memory errors can be broadly classified into **Heap Memory Errors** and **Stack Memory Errors**. Some of the challenging memory errors are:

- **Invalid Memory Access in heap and stack**
- **Memory leak**
- **Mismatched Allocation/Deallocation**
- **Missing Allocation**
- **Uninitialized Memory Access in heap and stack**
- **Cross Stack Access**

1)Invalid Memory Access

This error occurs when a read or write instruction references unallocated or deallocated memory.

```
char *pStr = (char*) malloc(25);
free(pStr);
strcpy(pStr, .parallel programming.); // Invalid write to deallocated memory in heap
```

2)Memory leaks

Memory leaks occur when memory is allocated but not released. If such leaks happen often enough and frequently enough, the leaks will eventually cause the application to run out of memory resulting in a premature termination (gracefully or as a crash).

```
char *pStr = (char*) malloc(512);
return;
```

3)Mismatched Allocation/Deallocation

This error occurs when a deallocation is attempted with a function that is not the logical counterpart of the allocation function used.

```
char *s = (char*) malloc(5);
delete s;
```

To avoid mismatched allocation/deallocation, ensure that the right deallocator is called. In C++, new[] is used for memory allocation and delete[] for freeing up. In C, malloc(), calloc() and realloc() functions are used for allocating memory while the free() function is used for freeing up allocated memory. Similarly, there are APIs in Windows programming to allocate and free memory.

4)Missing allocation This error occurs when freeing memory which has already been freed. This is also called "repeated free" or "double free". Example: char* pStr = (char*) malloc(20);
free(pStr);

free(pStr); // results in an invalid deallocation
Uninitialized Memory Access
This type of memory error will occur when an uninitialized variable is read in your application.

```
char *pStr = (char*) malloc(512);
char c = pStr[0]; // the contents of pStr were not initialized
void func()
{
int a;
int b = a * 4; // uninitialized read of variable a
}
```

To avoid this type of memory error, always initialize variables before using them.

5)Cross Stack Access

This occurs when a thread accesses stack memory of a different thread.

```
main()
{
int *p;
-------
CreateThread(., thread #1, .); // Stack Owned
CreateThread(., thread #2, .);
-------
}
Thread #1
{
int q[1024];
p = q;
q[0] = 1;
}
Thread #2
{
*p = 2; // Stack Cross Accessed
}
```

One of the easiest ways to avoid this error is to avoid saving stack addresses to global variables.

Q1) Create a new class named City that will have two member variables CityName (char[20]), and DistFromKtm (float).Add member functions to set and retrieve the CityName and DistanceFromKtm separately. Add new member function AddDistance that takes two arguments of class City and returns the sum of DistFromKtm of two arguments. In the main function, Initialize three city objects .Set the first and second City to be pokhara and Dhangadi.

Display the sum ofDistFromKtm of Pokhara and Dhangadi calling AddDistance function of third City object. [PU: 2010 Spring]

Ans:

```
#include<iostream>
 #include<string.h>
 using namespace std;

class City {
 private:
char CityName[20];
float DistFromKtm;
 public:
void setdata(char cname[],float d) {
strcpy(CityName,cname); DistFromKtm=d;
}
 void display() {
cout<<"City name="<<CityName<<endl;
 cout<<"Distance from ktm="<<DistFromKtm<<endl;
 }
 float AddDistance(City c1,City c2) {
 return (c1.DistFromKtm+c2.DistFromKtm);
}
 };

int main() {

City c1,c2,c3;

 c1.setdata("Pokhara",250);

c2.setdata("Dhangadi",150);

cout<<"Information of first city"<<endl;

c1.display();

 cout<<"Information of second city"<<endl;

c2.display();

cout<<"sum of DistFromKtm of Pokhara and Dhangadi="<<c3.AddDistance(c1,c2);

 return 0;
```

```
}
```

Q2)Design a soccer player class that includes three integer fields:a player's jersey number,number of goals,number of assists and necessary constructors to initialize the data members.Overload the > operator (greater than).One player is considered greater than another if the sum of goals plus assists is greater than that of the others.Create an array of 11 soccer players,then use the overloaded > operator to find the player who has the greatest total of goals plus assists.(PU 2015 fall)

Ans:

```
#include<string.h>
 #include<iostream>
 using namespace std;
 class splayer {
 char name [20];
 int goals ,jnum ,asst ;
 public :
 splayer(){
 goals=0;
jnum=0;
asst=0;
 strcpy(name," ");
 }
 splayer(char n[50],int g,int j,int a){
 strcpy(name,n);
 goals=g;
 jnum=j;
 asst=a;
 }
 void display (){
```

```
cout<<"name="<<name<<"\tgoals="<<goals<<"\tjoursey num="<<jnum<<"\tno.of asst=";
}
friend int operator >(splayer &s1,splayer &s2);
};
int operator >(splayer &s1,splayer &s2){
int sum1 =s1.goals +s1.asst ;
int sum2 =s2.goals +s2.asst ;
if(sum1 >sum2 ){
return 1;
}
else
return 0;
}

main (){
char nam[20];
int gl,jr,ast,max,i;
splayer greater;
splayer *s[11];
for(i=0;i<5;i++){
cout<<"<<enter player name, no.of goals,joursy num and num of asst..>>"<<endl;
cin>>nam>>gl>>jr>>ast;
s[i]=new splayer (nam,gl,jr,ast);

}
//displaying player info

cout<<"<<the given players detail is"<<endl;
for(int i=0;i<11;i++){
```

```
(
*s[i]).display();
}
int i=0;
max=i;
//for comparing every 2 players using greter than(>)operator sign
for( i=0;i<11;i++){
if(*s[i+1]>*s[max]) {
cout<<(i+1)<<"greater than"<<i<<endl;
max=i+1;
}
}
cout<<"the details of player with highest points(goals+assists) :" <<endl;
s[max]->display();
getch();
}
```

Q3) Define two class named Polar and Rectangle to represent points of polar and rectangle systems. Use conversion routine to convert from one system to another system.

Ans:

```
#include<iostream>
#include<math.h>
using namespace std;
class polar;
class rect{
        int x,y;
        public:
                rect(){
```

```
                x=0;
                y=0;
        }
        rect(int xx,int yy){
                x=xx;
                y=yy;
        }
        int getx(){
                return x;
        }
        int gety(){
                return y;
        }
        void display(){
                cout<<"xco="<<x<<endl<<"yco="<<y<<endl;
        }
rect(polar p);
};
class polar{
        int r,a;
        public:
                polar(){
                        r=0;
                        a=0;
                }
                polar(int rr,int aa){
                        a=aa;
```

```
                r=rr;
        }
        int getr(){
                return r;
        }
        int geta(){
                return a;
        }
        polar(rect re){
                r=sqrt(re.getx()*re.getx()+re.gety()*re.gety());
                a=atan(re.gety()/re.getx());

        }
        void display(){
                cout<<"radius="<<r<<endl<<"angle="<<a<<endl;
        }
};
        rect::rect(polar p){
                x=p.getr()*cos(p.geta());
                y=p.getr()*sin(p.geta());
        }
main(){
        rect r(10,8);
        r.display();
        polar p;
        p=r;
        p.display();
```

```
        polar p1(12,23);
        p1.display();
        rect r1;
        r1=p1;
        r1.display();
}
```

Q4)WAP to add two complex numbers using dynamic constructor.

Ans:

```
#include<iostream>
using namespace std;
class complex{
        int *r,*i;
        public:
                complex(){
                        r=new int(0);
                        i=new int(0);
                 }
                 complex(int re,int im){
                        r=new int(re);
                        i=new int(im);
                 }
                 void display(){
                        cout<<"real="<<*r<<"imag="<<*i;
                 }
        void add(complex c1,complex c2){
```

```
                *r=*c1.r+*c2.r;
                *i=*c1.i+*c2.i;
        }
};
main(){
    complex c1(1,2);
    c1.display();
    complex c2(3,4);
     c2.display();
    complex c;
    c.add(c1,c2);
    c.display();

}
```

Q5) WAP in c++ to calculate simple interest from given principal,time and rate .Set the rate to 15% as default argument when rate is not provided.

Ans:

```
#include<iostream>
 using namespace std;

class simple_interest{
        private:
        float p,t,r;
        public:
                        void interest(float principal, float time , float rate=0.15);
 };


 void simple_interest::interest(float principal, float time , float rate)
 {
        p=principal;
        t=time;
        r=rate;
```

```
            float si;
            si=(p*t*r)/100;
            cout<<"simple interest is :"<<si;
        }
        main()
        {
            float prin , ti ,ra;
            char c;
            simple_interest s1;
            cout<<"enter principal and time"<<endl;
            cin>>prin>>ti;
            cout<<"press y or Y if you want to give rate of interest";
            cin>>c;
            if(c=='y' || c=='Y'){
                cout<<"enter rate of interest";
                cin>>ra;
                s1.interest(prin,ti,ra);
            }


    else{

            s1.interest(prin,ti);

}

}
```

Q6)Create a class complex with two integer attributes real and imaginary.Include following responsibilities in the class:

- **default and parameterized constructors.**
- **Display method to display complex numbers in x+iy format where x=real and y=imaginary.**
- **Appropriate operator overload to realize multiplication of two complex numbers with * operator.**

  Ans:

```
#include<iostream>
using namespace std;
class complex{
        int x,y;
        public:
                complex(){
```

```
                x=0;
                y=0;
        }
        complex(int xx,int yy){
                x=xx;
                y=yy;
        }
        void display(){
                cout<<x<<"+i"<<y<<endl;
        }
complex operator *(complex b)
{
complex c;
c.x=((x*b.x)-(y*b.y));
c.y=((x*b.y)+(y*b.x));
return c;
}
};
main(){
        complex c1(1,2);
        complex c2(3,4);
        c1.display();
        c2.display();
        complex c3;
        c3=c1*c2;
        c3.display();

}
```

Q7)WAP to concatenate two string using composition.

```
Ans:#include<iostream>
using namespace std;
#include<conio.h>
#include<string.h>
class address{

        string adr;
  public:
        address(string a)
    {
```

```
        adr=a;
      }

      string getadr(){
          return adr;
      }

      void display();
  };
  void address::display(){
          cout<<"the address is: "<<adr<<endl;
  }

  class name{
          string nam;
    address adr1;
          public:
          name(string n,string a):adr1(a){
          nam=n;
      }

      void join();
          void display();

  };

  void name::join(){
          nam=nam+adr1.getadr();

    cout<<"name+address: "<<nam<<endl;
  }

  void name::display(){
          cout<<"the name is: "<<nam<<endl;
     adr1.display();
  }

int main(){
    string a="arun",b="ktm";
   name n1(a,b);
    n1.display();
    n1.join();
    getch();
    return 0;
```

```
}
```

Q8)WAP to calculate the volume by taking input as length,breadth and height in feet and inch.Total volume should be in cubic feet.Convert each single length brradth and height into feet and inch .Finally display the total volume.

Ans:

```
#include<iostream>
using namespace std;

class volume{
  private:
            float length, breadth, height;
            float inch, feet;
  public:
            void input_length();
            void input_breadth();
            void input_height();
            void display();
            void calculate();

};

void volume::input_length()
{
  cout<<"Enter length in foot and inch : "<<endl;
  cin>>feet>>inch;
  feet=feet+(inch/12);
  length = feet;
}
void volume::input_breadth()
{
  cout<<"Enter breadth in foot and inch : "<<endl;
  cin>>feet>>inch;
  feet=feet+(inch/12);
  breadth = feet;
}
void volume::input_height()
{
  cout<<"Enter height in foot and inch : "<<endl;
  cin>>feet>>inch;
  feet=feet+(inch/12);
  height = feet;
```

```
}

void volume::display()
{
 cout<<"Length = "<<length<<endl;
 cout<<"Breadth  = "<<breadth<<endl;
 cout<<"Height = "<<height<<endl;
}

void volume::calculate()

 cout<<"Volume of room = "<<length*breadth*height<<"cubic feet";
}
main()
{
 volume v;
 v.input_length();
 v.input_breadth();
 v.input_height();
 v.display();
 v.calculate();
 return 0;
}
```