Satyam Technical Paper 3

1. a. Different types of polymorphism?

1. b. types related by inheritance as polymorphic types because we can use many forms of a derived or base type interchangeably

2. only applies to ref or ptr to types related by inheritance.

3. Inheritance - lets us define classes that model relationships among types, sharing what is common and specializing only that which is inherently different

4. derived classes

    1. can use w/o change those operations that dont depend on the specifics of the derived type

    2. redefine those member functions that do depend on its type

    3. derived class may define additional members beyond those it inherits from its base class.

5. Dynamic Binding - lets us write programs that use objects of any type in an inheritance hierarchy w/o caring about the objects specific types

6. happens when a virtual function is called through a reference || ptr to a base class

7. The fact that a reference or ptr might refer to either a base or derived class object is the key to dynamic binding

8. calls to virtual functions made though a reference/ptr resolved @ runtime

9. the function that is called is the one defined by the actual type of the object to which ref/ ptr refers

2. How to implement virtual functions in C - keep function pointers in function and use those function ptrs to perform the operation

3. What are the different type of Storage classes?

    1. automatic storage: stack memory - static storage: for namespace scope objects and local statics

    2. free store: or heap for dynamically allocated objects == design patterns

4. What is a namespace?

    1. every name defined in a global scope must be unique w/in that scope

    2. name collisions: same name used in our own code or code supplied to us by indie producers == namespace pollution

    3. name clashing - namespace provides controlled mechanism for preventing name collisions

    4. allows us to group a set of global classes/obj/funcs

    5. in order to access variables from outside namespace have to use scope :: operator

    6. using namespace serves to assoc the present nesting level with a certain namespace so that objectand funcs of that namespace can be accessible directly as if they were defined in the global scope

5. Types of STL containers - containers are objects that store other objects and that has methods for accessing its elements - has iterator - vector

6. Difference between vector and array - -array: data structure used dto store a group of objects of the same type sequentially in memory - vector: container class from STL - holds objects of various types - resize, shrinks grows as elements added - bugs such as accessing out of bounds of an array are avoided

7. Write a program that will delete itself after execution. Int main(int argc, char **argv)

{ remove(argv[0]);return 0;}

8. What are inline functions?

    1. treated like macro definitions by C++ compiler

    2. meant to be used if there's a need to repetitively execute a small block if code which is smaller

    3. always evaluates every argument once

    4. defined in header file

    5. avoids function call overload because calling a function is slower than evaluating the equivalent expression

    6. it's a request to the compiler, the compiler can ignore the request

9. What is strstream? defines classes that support iostreams, array of char obj

10.Passing by ptr/val/refArg?

1. passing by val/refvoid c::f(int arg) ? by value arg is a new int existing only in function. Its initial value is copied from i. modifications to arg wont affect the I in the main function

2. void c::f(const int arg) ? by value (i.e. copied) the const keyword means that arg cant be changed, but even if it could it wouldnt affect the I in the main function

3. void c::f(int& arg) - -by reference, arg is an alias for I. no copying is done. More efficient than methods that use copy. Change in arg == change in I in the calling function

4. void c::f(const int& arg) - -by reference, int provided in main call cant be changed, read only. Combines safety with efficacy.

5. void c::f(const int& arg) const ? like previous but final const that in addition the function f cant change member variables of cArg passing using pointers

6. void c::f(int *arg) ? by reference changing *arg will change the I in the calling function.

7. void c::f(const int *arg) ? by reference but this time the I int in the main function cant be changed ? read only

8. void c::f(int * const arg) ? by reference the pointer arg cant be changed but what it points to (namely I of the calling function) can

9. void c::f(const int * const arg) by reference the pointer arg cant be changed neither can what it points to