Question 1. **Solution:**

#include <iostream>

using namespace std;

class MToy {//Moving Toy

public:

MToy() = default;

MToy(const MToy& obj){

cout << "MToy Copy Constructor" << endl;

}

MToy& operator= (const MToy& obj){

cout << "MToy Assignment operator" << endl;

return \*this;

}

~MToy(){}

};

class FToy {//Flying Toy

public:

FToy() = default;

FToy(const FToy& obj){

cout << "FToy Copy Constructor" << endl;

}

FToy& operator= (const FToy& obj){

cout << "FToy Assignment operator" << endl;

return \*this;

}

~FToy(){}

};

**//solution for MFtoy**

class MFToy:public MToy, public FToy {//Moving & Flying Toy

public:

MFToy() = default;

MFToy(const MFToy& obj):MToy(obj),FToy(obj){

cout << "MFToy Copy Constructor" << endl;

}

MFToy& operator= (const MFToy& obj){

MToy::operator = (obj); //explicit call to parent = operator

FToy::operator = (obj);

cout << "MFToy Assignment operator" << endl;

return \*this;

}

~MFToy(){}

};

int main()

{

/\*MFtoy ---- Moving & Flying Toy\*/

MFToy mf1, mf2;

MFToy mf3 = mf1;

mf2 = mf3;

return 0; }

Question 2. **Solution:**

#include <iostream>

using namespace std;

class A {

public:

virtual void print(){

/\* One word change i.e., adding void keyword here \*/

cout << "Class A\n";

}

};

class B: public A {

public:

void print(){

cout << "Class B\n";

}

};

class C:public B {

public:

void print(){

cout << "Class C\n";

}

};

void doPrint(A &a){ //doPrint() Function

a.print();

}

int main()

{

A aa; B bb; C cc;

doPrint(aa);

doPrint(bb);

doPrint(cc);

return 0; }

Question 3. **Solution:**

#include <iostream>

using namespace std;

class AA {

public:

void print(){

cout << "Class AA\n";

}

};

class BB{

public:

void print(){

cout << "Class BB\n";

}

};

class CC:public AA, public BB {

public:

void print(){

/\*Required Function\*/

BB::print();

AA::print();

}

};

int main()

{

CC cc;

cc.print();

return 0; }

Question 4. **Solution:**

class DynamicArray {

int\* arr;

int size; //number of elements in arr

int capacity; //number of elements that arr can hold before regrowing

public:

//public interface

int insert(int pos, int count, int value){

int\* tempArr;

int tempSize = size;

tempArr = new int[tempSize];

for(int i=0; i< tempSize; i++){

tempArr[i] = arr[i];

}

delete[] arr;

capacity = size + count;

arr = new int[capacity]; //regrowing

int start\_pos = pos;

int end\_pos = pos+count;

int index = 0;

int i = 0;

for(i=0; i<start\_pos; i++){

arr[index++] = tempArr[i]; //copying

}

for(index=start\_pos; index<end\_pos; index++){

arr[index] = value; //copying required value on required location

}

for(index=end\_pos; i<tempSize; i++){

arr[index++] = tempArr[i]; //copying remain values

}

size = capacity;

return end\_pos;

}

};