**Quiz 2 Answer Guide:**

**1:**

**Answer:-**

One of the important concepts of OOP is data hiding, i.e., a nonmember function cannot access an object's private or protected data. But, sometimes this restriction may force programmer to write long and complex codes. So, there is mechanism built in C++ programming to access private or protected data from non-member functions. This is done using a friend function or/and a friend class.

If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

**2:**

**Answer:-**

When a parameter is passed by reference, the caller and the callee use the same variable for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller's variable.

When a parameter is passed by value, the caller and callee have two independent variables with the same value. If the callee modifies the parameter variable, the effect is not visible to the caller. Hence when we want the object to be independent per function, we should always prefer call by value.

Fun Fact - Java have deprecated call by reference and only allows call by value.

**3.**

**Answer :-**

#include <iostream>

using namespace std;

class CMyClass

{

public:

static int m\_i;

};

int CMyClass::m\_i = 0;

CMyClass myObject1;

CMyClass myObject2;

int main()

{

cout << myObject1.m\_i << endl; //Answer is =>0

cout << myObject2.m\_i << endl; //Answer is =>0

myObject1.m\_i=1;

cout << myObject1.m\_i << endl; //Answer is =>1

cout << myObject2.m\_i << endl; //Answer is =>1

myObject1.m\_i=2;

cout << myObject1.m\_i << endl; //Answer is =>2

cout << myObject2.m\_i << endl; //Answer is =>2

}

**4.**

**Answer:-**

// FILE: bag1.cxx

// CLASS IMPLEMENTED: bag (see bag1.h for documentation)

// INVARIANT for the bag class:

// 1. The number of items in the bag is in the member variable used;

// 2. For an empty bag, we do not care what is stored in any of data; for a

// non-empty bag the items in the bag are stored in data[0] through

// data[used-1], and we don't care what's in the rest of data.

#include <algorithm> // Provides copy function

#include <cassert> // Provides assert function

#include "bag1.h"

using namespace std;

namespace main\_savitch\_3

{

const bag::size\_type bag::CAPACITY;

bag::size\_type bag::erase(const value\_type& target)

{

size\_type index = 0;

size\_type many\_removed = 0;

while (index < used)

{

if (data[index] == target)

{

--used;

data[index] = data[used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bool bag::erase\_one(const value\_type& target)

{

size\_type index; // The location of target in the data array

// First, set index to the location of target in the data array,

// which could be as small as 0 or as large as used-1. If target is not

// in the array, then index will be set equal to used.

index = 0;

while ((index < used) && (data[index] != target))

++index;

if (index == used)

return false; // target isn’t in the bag, so no work to do.

// When execution reaches here, target is in the bag at data[index].

// So, reduce used by 1 and copy the last item onto data[index].

--used;

data[index] = data[used];

return true;

}

void bag::insert(const value\_type& entry)

// Library facilities used: cassert

{

assert(size( ) < CAPACITY);

data[used] = entry;

++used;

}

void bag::operator +=(const bag& addend)

// Library facilities used: algorithm, cassert

{

assert(size( ) + addend.size( ) <= CAPACITY);

copy(addend.data, addend.data + addend.used, data + used);

used += addend.used;

}

bag::size\_type bag::count(const value\_type& target) const

{

size\_type answer;

size\_type i;

answer = 0;

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

if (target == data[i])

++answer;

return answer;

}

bag operator +(const bag& b1, const bag& b2)

// Library facilities used: cassert

{

bag answer;

assert(b1.size( ) + b2.size( ) <= bag::CAPACITY);

answer += b1;

answer += b2;

return answer;

}

}