CMP-142 Object Oriented Programming
BS Fall 2014
Quiz 06 KEY

RollNo:

Issue Date: 08-May-2015
Time: 50 min
Marks: 21

Objective:

- The purpose of this quiz is to focus on the very basic fundamental concepts learned so far in previous lectures.

Question No. 1: (2)

Name at least two operators, which cannot be overloaded in C++.

Dot operator
Ternary conditional operator

Question No. 2: (3,2,2)

Consider the following classes for keeping track of vote totals for candidates in an election:

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

class Candidate
{
private:
	CString name;
	int votes;
public:
	Candidate(CString N=" ");
	CString getName() const;
	int getVotes() const;
	void addVote();
};
Candidate::Candidate(CString n): name(n)
{
	votes = 0;
}
CString Candidate::getName() const
{
	return name;
}
int Candidate::getVotes() const
{
	return votes;
}
void Candidate::addVote()
{
	votes++;
}
```

```
class VotingMachine
{
private:
		int numCand;
		Candidate * total;
public:
   VotingMachine(const CString p[], const int N);
   bool countVote(const CString& Name);
   Candidate report(int ind) const;
   ~VotingMachine();
};
VotingMachine::VotingMachine(const CString p[], int N)
{
   total = 0;
   numCand = 0;
   if (N>0)
   {
		numCand = N;
		total = new Candidate[numCand];
		for (int i =0; i<numCand; i++)
			total[i] = Candidate(p[i]);
   }
}
Candidate VotingMachine::report( int ind ) const
{
   if (ind<0 || ind >= numCand )
		return Candidate("none");
   return total[ind];
}


VotingMachine::~VotingMachine()
{
   delete [] total;
```

For the next three questions, Assume that a VotingMachine object named 'vm' has been declared and properly initialized.

Part A). Calling the function below with the argument passed 'vm' will result in an unfortunate side effect, even though the body of the function is correctly implemented. Describe the side effect briefly but clearly.

```
//function to print a table of the results from a particular voting machine object.

    void results(const VotingMachine v)
    {
            int ind =0;
            Candidate curr = v.report(ind);
            CString currName = curr.getName();
            cout<<"Candidate Votes"<<endl;
            while(! (currName=="none") )
            {
                    curr.getName().display();
                    cout<<<<"\t\t"<<curr.getVotes()<<endl;
                    ind++;
                    curr = v.report(ind);
                    currName = curr.getName();
            }
    }
```

The Candidate array of VotingMachine will be deleted at the end of the execution of the function results(), when the local object v is destructed. [The question is about what happens, not why it happens? The why is that the VirtualMachine class doesn't provide a deep copy constructor, so the object V is a shallow copy of VotingMachine and their pointers have the same value, so the two objects share the same array.]

Part B). Which of the following terms best characterizes the side effect (NOT possible consequences of statements following the function call) referred to in **Part-A**

A). Memory leak
B). memory corruption
C). A dangling pointer
D). access violation
E). None of the above

Part C). Which of the following should be done, specifically in order to eliminate the side effect referred to Part-B?

A). Add a deep copy ctor to VotingMachine class
B). Add a deep copy ctor to Candidate class
C). Add a deep assignment operator to Candidate class
D). All of the above
E). A & B only
F). C & D only
G). None of the above

Question No. 3: (2,3,4,2,1)

I hope you remember the class 'Array' and class 'Matrix' discussed in lecture/lab. Assume that all the definitions for member functions are given for both classes.

```
class Matrix
{
private:
        int * * data;
        int row;
        int col;

        isValidBounds( int , int );

public:
        Matrix ( )
        {
                data = 0;
                row = col = 0;
        }
        Matrix ( int r, int c );
        Matrix ( const Matrix & );
        Matrix & operator = ( const Matrix & );
        .
        .
        .
        //many other functions
}
```

Assume that in class 'Matrix', data member: 'int * * data' is replaced with 'Array * * data'. Considering this change: give definition of the following member functions in 'Matrix'.

- Matrix (int r, int c);
- Matrix & operator = (const Matrix &);
- Operator [] *//give appropriate prototype and definition yourself*
- Array operator (int rowNumber);

  It returns a copy of the matrix row mentioned in rowNumber.
  e.g;
  Matrix m(4,3);
  Array a = m(2);//it returns a copy of row number 2 of object 'm'.

- How would you define 'operator []' for constant objects of Matrix. Because for constant Matrix objects: matrix elements must not be changed.

```
class Matrix
{
private:
        Array * * data;
        int row;
        int col;

        bool isValidBounds( int i, int j)
        {
          return i>=0 && i<row && j>=0 && j<col;
        }

public:

        Matrix ( )
        {
          data = 0;
          row = col = 0;
        }
        Matrix ( int r, int c )
        {
          if (r<=0 || c<=0)
          {
               data=0;
               row = col = 0;
               return;
          }
          row=r;
          col=c;
          data = new Array*[row];
          for ( int i=0; i<row; i++ )
          {
               data[i] = new Array(col);
          }
        }
        Matrix ( const Matrix & ref)
        {
            row = ref.row;
            col = ref.col;
            if (ref.data==0)
            {
                data=0; return;
            }
            data = new Array*[row];
            for ( int i=0; i<row; i++ )
            {
                data[i] = new Array(*(ref.data[i]));
            }
        }
        Matrix & operator = ( const Matrix & ref)
        {
            if ( this==&ref )
                return *this;
            row = ref.row;
            col = ref.col;
            if (ref.data==0)
            {
                data=0;
                return *this;
            }
            data = new Array* [row];

            for ( int i=0; i<row; i++ )
            {
                data[i] = new Array(*(ref.data[i]));
            }
            return *this;
        }
        Array operator () (int rowNumber)
        {
            if (!(rowNumber>=0 && rowNumber<row))
                exit(0);
            return *data[rowNumber];
        }
        Array & operator [] (int i)
        {
            if (!(i>=0 && i<row))
                exit(0);
            return *data[i];
        }
        const Array & operator [] (int i) const
        {
            if (!(i>=0 && i<row))
                exit(0);
            return *data[i];
        }
};

int main()
{
        Matrix m(2,3);
        for ( int i=0; i<2; i++)
        {
            for ( int j=0; j<3; j++)
            {
                m[i][j]=i+j;
            }
        }
        for ( int i=0; i<2; i++)
        {
            for ( int j=0; j<3; j++)
            {
                cout<<m[i][j]<<endl;
            }
        }

        return 1;
}
```