Inheritance in C++

Question 1 (from Ford & Topp)

Consider the following inheritance hierarchy:

```cpp
class baseCL {
public:
    baseCL();
    void demoFunc();

private:
    int m;

protected:
    int n;
};

class derivedCL: public baseCL {
public:
    derivedCL();
    void demoFunc();

private:
    int r;
};
```

(a) Which of the data members m, n, and r can be accessed by a member function in the derived class?

(b) Which of the data members m, n, and r can be accessed by a member function in the base class?

(c) Consider the declarations:

```cpp
    baseCL bObj;
    derivedCL dObj;
```

Which of the objects bObj and dObj can execute demoFunc() in the base class? If valid, give the C++ statement that provides the function call.

Which of the objects bObj and dObj can execute demoFunc() in the derived class? If valid, give the C++ statement that provides the function call.

Question 2 (from Ford & Topp)

The given program illustrates the order in which classes in an inheritance hierarchy make constructor and destructor calls. What is the output of the program?

```cpp
#include <iostream>

using namespace std;
```
class baseCL {
public:
    baseCL(){cout << "baseCL constructor" << endl;}
    ~baseCL(){cout << "baseCL destructor" << endl;}
};

class derivedCL: public baseCL {
public:
    derivedCL(){cout << "derivedCL constructor" << endl;}
    ~derivedCL(){cout << "derivedCL destructor" << endl;}
};

int main(){
    baseCL bObj;
    derivedCL dObj;

    return 0;
}

Question 3

Write a generic class, named Queue, in C++ for the queue type that uses a linked list to store the elements. The Queue class has a member variable, named head, that references the first node of the list, a member variable, named tail, that references the last node of the list, and a member variable, named size, that stores the number of elements in the queue. The Queue class provides all of the methods of the STL queue class, including push, pop, front, and empty. The Node class is defined as follows.

```cpp
template <typename T>
class Node {
public:
    T nodeValue;
    Node<T> *next;
    Node (const T& item, Node<T> *ptr = NULL): nodeValue(item), next(ptr) {};
};
```

Write a class, named DerivedQueue, which extends Queue by providing a method named emergency_push that inserts an element at the front of the queue.

Question 4 (from Ford & Topp)

Use the employee hierarchy (see below) and the following statements for this problem:

```cpp
employee boss("Mr. Boss", "111-222-333"), *p;
salaryEmployee sEmp("Steve Howard","896-54-3217",3330.00), *q = &sEmp;
```
hourlyEmployee hEmp("Johns Ross","896-54-3217",7.50,40), *r = &hEmp;

p = &sEmp;

Indicate the version of displayEmployeeInfo() that is executed by each of the following function calls:

r->displayEmployeeInfo();
q->displayEmployeeInfo();
q->employee::displayEmployeeInfo();
p->displayEmployeeInfo();

// base class for all employees
class employee
{
public:
    // constructor
    employee(const string& name, const string& ssn) :
        empName(name), empSSN(ssn)
    {}

    // output basic employee information
    virtual void displayEmployeeInfo() const
    {
        cout << "Name: " << empName << endl;
        cout << "Social Security Number: " << empSSN << endl;
    }

    // function with this prototype will exist in each derived class
    virtual void payrollCheck() const
    {}

protected:
    // maintain an employee’s name and social
    // security number
    string empName;
    string empSSN;
};

// salaried employee "is an" employee with a monthly salary
class salaryEmployee : public employee
{
public:
    // initialize Employee attributes and monthly salary
    salaryEmployee(const string& name, 
                   const string& ssn, double sal):
        employee(name,ssn),salary(sal)
    {}
void setSalary(double sal)
{ salary = sal; }

void displayEmployeeInfo() const
{
    employee::displayEmployeeInfo();
    cout << "Status: salaried employee" << endl;
    cout << "Salary per week $" << setreal(1,2)
         << salary << endl;
}

virtual void payrollCheck() const
{
    cout << "Pay " << empName << " (" << empSSN
         << ") $" << setreal(1,2) << salary << endl;
}

private:

class hourlyEmployee : public employee
{
public:
    // initialize Employee attributes, hourly pay rate
    // and hours worked
    hourlyEmployee(const string& name, const string& ssn,
                   double hp, double hw) : employee(name,ssn),
                           hourlyPay(hp), hoursWorked(hw)
{}

    // update the hourly pay and hours worked
    void setHourlyPay(double hp)
    { hourlyPay = hp; }

    void setHoursWorked(double hw)
    { hoursWorked = hw; }
}
// call displayEmployeeInfo from base class and output info
// on hourly rate and scheduled hours
void displayEmployeeInfo() const
{
    employee::displayEmployeeInfo();
    cout << "Status: hourly employee" << endl;
    // cout << "Payrate: $" << setreal(1,2)
    cout << "Payrate: $" << hourlyPay << " per hour" << endl;
    cout << "Work schedule (hours per week) " << hoursWorked
        << endl;
}

virtual void payrollCheck() const
{
    cout << "Pay " << empName << " (" << empSSN << ") $"
    // << setreal(1,2) << (hourlyPay * hoursWorked) << endl;
    // << (hourlyPay * hoursWorked) << endl;
}

private:
// pay based on hourly pay and hours worked
double hourlyPay;
double hoursWorked;
};