Advanced Programming Concepts with C++

CSI 2372

Tutorial # 7

Midterm Exam solutions

uOttawa

Ahmedou Jreivine

PART A: SHORT QUESTIONS (8 MARKS)

1. Given the following declaration [1]

```
struct Line {
    const double d_len;
    Line( double len ) : d_len{ len } {}
};
struct TrainLine : public Line {
    TrainLine( double len );
};
```

Implement the above declared constructor for Trainline such that Line:d_len is initialized (!) with the argument len.

Answer:

- `TrainLine::TrainLine( double len ) : Line( len ) {}`

PART A: SHORT QUESTIONS (8 MARKS)

2. What is printed be the following program? [1]

```
#include <typeinfo>
class A {};
class B : public A {};
class C : public A {};
... // omitted
B b;
A& a = b;
if ( &a == nullptr ) cout << "&a is Nullptr!" << endl;
C* c = dynamic_cast<C*>(&a);
if ( c == nullptr ) cout << "c is Nullptr!" << endl; // is printed.
try {
C* cEx = dynamic_cast<C*>(&a);
} catch ( std::bad_cast exp) {
cout << "cEx is Nullptr!" << endl;
}
```

uOttawa

PART A: SHORT QUESTIONS (8 MARKS)

3. What is printed by the following? [1]

```
struct Box {
    static bool isSquare;
    Box() {
        if (isSquare) cout << "square" << endl;
        else cout << "rectangle" << endl;
        isSquare = !isSquare;
    }
}
bool Box::isSquare = true;
int main() {
    Box rsa, rsb, rsc; // "square" , "rectangle" and "square"
}
```

PART A: SHORT QUESTIONS (8 MARKS)

4. What is printed by the following? [1]

```
struct Animal {
    int d_nLegs=4;
    Animal() {}
    Animal(int nLegs) : d_nLegs{nLegs} {}
    Animal(bool b, int nFront, int nHind) :
    d_nLegs{nFront+nHind} {}
};
struct Insect : public Animal {
    Insect() {}
    Insect(int nLegs) : Animal{nLeg} {}
    Insect(bool b, int nFront=3, int nHind=3 ) {}
};
int main() {
    Insect ia, ib{16}, ic{true};
    cout << ia.d_nLegs << " " << ib.d_nLegs << " " << ic.d_nLegs;  // 4 16 4
    return 0;
}
```

PART A: SHORT QUESTIONS (8 MARKS)

5. Call the function out() on the object with ptr [1]

```
struct IO {
    …
    int out() { }
};
int main() {
    IO* ptr = new IO();
    ptr->out();
    //or
    (*ptr).out();
    return 0;
}
```

PART A: SHORT QUESTIONS (8 MARKS)

```
class Bird {
public:
    virtual void name() { cout << "Bird" << " "; }
    void numLegs() { cout << 2 << " "; }
    virtual void flight() = 0;
};
class Emu : public Bird {
    public:
    void name() { cout << "Emu" << " "; }
    void numLegs() { cout << 3 << " "; }
    void flight() override { cout << "no" << " "; }
};
int main() {
    Emu e;
    Bird& b = e;
    e.name(); e.numLegs(); e.flight(); cout << endl;
    b.name(); b.numLegs(); b.flight();
    cout << endl;
    return 0;
}
```

Output:

Emu 3 no
Emu 2 no

PART B: Short Programs (7 MARKS)

```
#include <iostream>
#include <list>
#include <memory>
class Pizza {
    int d_nTop;
    float d_price;
public:
    Pizza() { std::cout << "Default ctor" << std::endl;}
    Pizza(int nTop, float price = 10.0f) : d_nTop{ nTop }, d_price{ price } { std::cout << "Full ctor" <<
    std::endl; }
    Pizza(const Pizza& p) : d_nTop{ p.d_nTop },d_price{ p.d_price } { std::cout << "Copy ctor" << std::endl; }
    Pizza& operator=(const Pizza& p) {
        d_nTop = p.d_nTop;
        d_price = p.d_price;
        std::cout << "Assignment" << std::endl;;
        return *this;
    }
    float getPrice() { return d_price; }
    void setPrice(float price) { d_price = price; return; }
    float getTop() { return d_nTop; }
    void setTop(int nTop) { d_nTop = nTop; return; }
};
```

PART B: Short Programs (7 MARKS)

```
Pizza deliver(Pizza& p) {  p.setPrice(p.getPrice() + 5.0f); return p; }
Pizza* extra(Pizza* p) { p->setTop(p->getTop() + 1); return p; }

int main(){
    std::cout << "Question 1.1" << std::endl;
    Pizza cheese(2, 11.0f);                 // Full ctor
    std::cout << "Question 1.2" << std::endl;
    Pizza plain;                            // Default ctot
    std::cout << "Question 1.3" << std::endl;
    Pizza doubleCheese(cheese);             //Copy ctor
    std::cout << "Question 1.4" << std::endl;
    extra(&doubleCheese);                   // 3
    std::cout << doubleCheese.getTop() << std::endl;
    std::cout << "Question 1.5" << std::endl;
    cheese = deliver(cheese);               //Assignment
    std::cout << "Question 1.6" << std::endl;
    Pizza* p = extra(extra(&plain));
    std::cout << p->getTop() << " " << plain.getTop() << std::endl;     //2     2
    return 0;
}
```

PART B: Short Programs (7 MARKS)

2. Consider the classes Computer and Laptop defined as follows:

```
#include<iostream>
#include<string>
using namespace std;
class Computer {
    string d_name;
    int d_noProcessors;
    float d_clockRate;
public:
    Computer(string _name, int _noProcessors = 1, float _clockRate = 1.8)
    : d_name{ _name }, d_noProcessors{ _noProcessors },d_clockRate(_clockRate) {}
    virtual ~Computer() {}
    void print() {
        cout << d_name << " with " << d_noProcessors << " at " << d_clockRate << endl;
    }
            const string& getName() { return d_name; }
            virtual void update() = 0;
};
```

PART B: Short Programs (7 MARKS)

```
class Laptop : public Computer {
    int d_graphicsDriver;
public:
    Laptop(string _name = "Dell") :Computer{ _name, 1, 1.2 },
    d_graphicsDriver{ 1 } {}
    void print() {
        cout << "Laptop: ";
        Computer::print();
        cout << "Graphics version: " << d_graphicsDriver << endl;
    }
    void update() {
        if (getName() != "Dell")
        ++d_graphicsDriver;
    }
};
```

PART B: Short Programs (7 MARKS)

```
int main() {
        Computer* lt = new Laptop("Acer");
        Laptop tmp;
        Computer& ct = tmp;
        lt->update();
        lt->print();
        ct.update();
        ct.print();
        tmp.print();
        return 0;
}
```

a. What will be printed:

Acer with 1 at 1.2
Dell with 1 at 1.2
Laptop: Dell with 1 at 1.2
Graphics version: 1

PART B: Short Programs (7 MARKS)

b. Assume we change the line

class Laptop : public Computer to

class Laptop : private Computer

Will the main program still compile and run? Very briefly explain why [2].

Anwer:

No, because the members of Computer with be inherited as private in Laptop. Therefore the Laptop objects cannot access the inherited Computer members.

PART C: PROGRAMMING QUESTIONS (8 MARKS)

For the following linked list implementation, a deep copy strategy is needed.

```
struct ListBox {
    int d_id;
    ListBox* d_prev{nullptr};
    ListBox* d_next{nullptr};
public:
    ListBox( int id ) : d_id{id} {};
    ListBox* last() { if ( d_next == nullptr ) return this; return d_next->last();} // go to the last box
    ListBox* find(int count) {                    // find forward -- count must be positive or 0
        if (count <= 0) return this;
        if ( d_next == nullptr ) return nullptr;
        return d_next->find( --count );
    }
    ListBox* findReverse(int count) {             // find reverse -- count must be negative or 0
        if (count >= 0) return this;
        if ( d_prev == nullptr ) return nullptr;
        return d_prev->findReverse(--count);
    }
    void insert( ListBox* newB ) {                // insert the supplied box after this
        newB->d_next = d_next;
        newB->d_prev = this;
        d_next = newB;
        if ( newB->d_next != nullptr ) {          // guard for last box
            newB->d_next->d_prev = newB;
        }
    }
```

PART C: PROGRAMMING QUESTIONS (8 MARKS)

1. Implement the destructor for ListBox [2].

```
~ListBox() {        // delete this and everything after it
    if (d_prev != nullptr) d_prev->d_next = nullptr;
    if (d_next != nullptr) delete d_next;
}
```

2. Implement a deep copy constructor for ListBox [3].

```
// copy this and eveything after it
ListBox(const ListBox& oLB) : d_boxId{ oLB.d_boxId }{
    if (oLB.d_next != nullptr) {
        d_next = new ListBox{ *oLB.d_next };
        d_next->d_prev = this;
    }
}
```

PART C: PROGRAMMING QUESTIONS (8 MARKS)

3. Implement the assignment operator using a deep copy strategy for ListBox [3].

```
ListBox& operator=(const ListBox& oLB){        //assign this and eveything after it
    if (&oLB != this) {
        d_boxId = oLB.d_boxId;
        d_prev = nullptr;
        // first we delete our own list
        if (d_next != nullptr) delete d_next;
        d_next = nullptr;
        if (oLB.d_next != nullptr) {                    // build our own list
            d_next = new ListBox{ *oLB.d_next };
            d_next->d_prev = this;
        }
    }
}
```