COP4530 – C++ Basics Programming Assignment 2

Section 1: Inheritance and Classes (25 pts)

1. Generalized Hierarchy Code

- Person
  - Student
    - UndergradStudent
    - GradStudent
  - Faculty
    - Professor
    - Instructor

```cpp
#include <string>
#include <vector>

class Person {
protected:
    std::string name;
    int age;

public:
    Person(const std::string& name, int age)
        : name(name), age(age) {}
};

class Student : public Person {
protected:
    std::string studentId;
    double gpa;

public:
    Student(const std::string& name,
            int age,
            const std::string& studentId,
            double gpa)
        : Person(name, age),
          studentId(studentId),
          gpa(gpa) {}
};

class UndergradStudent : public Student {
    std::string major;

public:
    UndergradStudent(const std::string& name,
                     int age,
                     const std::string& studentId,
                     double gpa,
                     const std::string& major)
        : Student(name, age, studentId, gpa),
          major(major) {}
};
```

```cpp
class GradStudent : public Student {
    std::string researchTopic;

public:
    GradStudent(const std::string& name,
                int age,
                const std::string& studentId,
                double gpa,
                const std::string& researchTopic)
        : Student(name, age, studentId, gpa),
          researchTopic(researchTopic) {}
};

class Faculty : public Person {
protected:
    std::string facultyId;
    std::string department;

public:
    Faculty(const std::string& name,
            int age,
            const std::string& facultyId,
            const std::string& department)
        : Person(name, age),
          facultyId(facultyId),
          department(department) {}
};

class Professor : public Faculty {
    std::string researchTopic;
    std::vector<std::string> classesTaught;

public:
    Professor(const std::string& name,
              int age,
              const std::string& facultyId,
              const std::string& department,
              const std::string& researchTopic,
              const std::vector<std::string>& classesTaught)
        : Faculty(name, age, facultyId, department),
          researchTopic(researchTopic),
          classesTaught(classesTaught) {}
};

class Instructor : public Faculty {
    std::vector<std::string> classesTaught;

public:
    Instructor(const std::string& name,
               int age,
               const std::string& facultyId,
               const std::string& department,
               const std::vector<std::string>& classesTaught)
        : Faculty(name, age, facultyId, department),
          classesTaught(classesTaught) {}
};
```

2. Inheritance Structure

`Person` is the root; `Student` and `Faculty` derive directly from it. `UndergradStudent` and `GradStudent` both extend `Student` with a `major` or `researchTopic`, respectively. `Professor` and `Instructor` both extend `Faculty`, differing only in the `classesTaught` they manage and their research interests.

3. Member Variables

Each class contributes its own state:

- `Person`: `name`, `age`
  - `Student`: `studentId`, `gpa`
    - `UndergradStudent`: `major`
    - `GradStudent`: `researchTopic`
  - `Faculty`: `facultyId`, `department`
    - `Professor`: `researchTopic`, `classesTaught`
    - `Instructor`: `classesTaught`

Section 2: Dynamic Casting (25 pts)

```cpp
class Object { public: virtual void printMe() = 0; };
class Place : public Object { public: virtual void printMe() { cout << "Buy it.\n"; } };
class Region : public Place { public: virtual void printMe() { cout << "Box it.\n"; } };
class State : public Region { public: virtual void printMe() { cout << "Ship it.\n"; } };
class Maryland : public State { public: virtual void printMe() { cout << "Read it.\n"; } };

int main() {
    Region* mid = new State;
    State* md = new Maryland;
    Object* obj = new Place;
    Place* usa = new Region;
    md->printMe();
    mid->printMe();
    (dynamic_cast<Place*>(obj))->printMe();
    obj = md;
    (dynamic_cast<Maryland*>(obj))->printMe();
    obj = usa;
    (dynamic_cast<Place*>(obj))->printMe();
    usa = md;
    (dynamic_cast<Place*>(usa))->printMe();
    return EXIT_SUCCESS;
}
```

Execution output reasoning:

1. md points to a Maryland object, so Maryland::printMe() executes, printing "Read it."
2. mid points to a State object, so State::printMe() executes, printing "Ship it."
3. obj was initialized with new Place, so the Place version executes.
4. Reassigning obj to md changes its underlying target object to Maryland. The dynamic_cast succeeds, and the Maryland version runs.

5. usa originally pointed to a Region. When obj is assigned to usa, it also points to that Region object. Even though you cast the pointer to Place*, the virtual table ensures the Region method executes.
6. usa is reassigned to point to md (Maryland). Casting a pointer to a Maryland object up to a Place* doesn't change the object's true identity, so Maryland::printMe() executes.

Section 3: Generic Types (25 pts)

Sample `Pair` class demonstration:

```cpp
#include <iostream>
#include <string>

template <typename T, typename U>
class Pair {
public:
    T first;
    U second;
    Pair(const T& first, const U& second) : first(first), second(second) {}
    void print() const {
        std::cout << "(" << first << ", " << second << ")\n";
    }
};

int main() {
    Pair<int, std::string> p1(1, "one");
    Pair<float, long> p2(3.14f, 42L);
    Pair<char, bool> p3('A', true);
    Pair<double, double> p4(2.7, 7.1);
    Pair<std::string, std::string> p5("hello", "world");

    p1.print();
    p2.print();
    p3.print();
    p4.print();
    p5.print();
    return 0;
}
```

Section 4: Linked List Programming (25 pts)

Recursive singly-linked list reversal idea:

```cpp
Node* reverse(Node* head) {
    if (!head || !head->next) return head;
    Node* rest = reverse(head->next);
    head->next->next = head;
    head->next = nullptr;
    return rest;
}
```

The recursion unwinds from the tail, reassigning each `next` pointer to point backward and finally returning the new head.