CSED232: Object-Oriented Programming Final Exam

Deadline: June 27th, Saturday, 23:59pm

Note: 답안은 MS 워드나 한컴 한글 등으로 작성 후 PDF 파일로 변환 후 제출하시기 바랍니다. 본 시험은 Take-home exam 으로 교재 나 인터넷 상의 자료를 참고해서 풀어도 좋습니다. **그러나 다른 사람과의 상의는 금지합니다.** 답안은 마감 시한 이후에는 받지 않으니 꼭 마감시간 안에 LMS 를 통해 제출하시기 바랍니다. 문의사항은 조성현 교수에게 이메일 (s.cho@postech.ac.kr) 을 통해 연락 바랍니다.

1. 다음 개념들이 각각 무엇을 뜻하는지 한두줄 내외로 간략히 기술하시오. (각 4 점, 총 28 점)
   A. Pure virtual function
   B. Function overloading
   C. Function overriding
   D. Dynamic binding
   E. Abstract base class
   F. Generic programming
   G. Event-driven programming

2. C++ Standard Template Library 의 algorithm 들은 iterator 를 사용하도록 설계되어 있다. iterator 를 사용하는 것의 장점이 무엇인지에 대해 간략하고 명확하게 기술하시오. (4 점)

3. Has-a relationship 을 구현하기 위한 두 가지 대표적인 방법으로 containment 와 private inheritance 가 있다. Has-a relationship 이 무엇인지 기술하고, 또한 containment 와 private inheritance 의 장단점을 비교하시오. (8 점)

4. Private inheritance 와 protected inheritance 의 차이점을 자세히 기술하시오. (4 점)

5. 다음 코드를 실행시켰을 때 출력되는 내용은 무엇인가? 그리고 왜 그런 출력이 나오는가? 다음 코드를 실행시켰을 때 출력될 내용을 적고, 그 이유를 간단히 기술하시오. (8 점)

```
class Base {
public:
    Base(int n=0) : m_n(n) {}
    virtual ~Base() {}
    virtual void display() const {
        std::cout << "Base: " << m_n << std::endl;
    }
private:
    int m_n;
};

class Derived1 : public Base {
public:
    Derived1(int n=0) : Base(2*n), m_n(n) {}
    ~Derived1() {}
    void display() const {
        Base::display();
        std::cout << "Derived1: " << m_n << std::endl;
    }
private:
    int m_n;
};

class Derived2 : public Base {
public:
    Derived2(int n=0) : Base(3*n), m_n(n) {}
    ~Derived2() {}
    void display() const {
        Base::display();
        std::cout << "Derived2: " << m_n << std::endl;
    }
private:
    int m_n;
};

class DDerived : public Derived1, public Derived2
{
public:
    DDerived(int n, int m) : Derived1(n), Derived2(m) {}
    ~DDerived() {}

    void display() const {
        Derived1::display();
        Derived2::display();
    }
};

int main()
{
    std::unique_ptr<Derived1> pt(new DDerived(3,5));
    pt->display();
    return 0;
}
```

6. 다음은 재귀 호출을 이용하여 factorial 을 계산하는 코드이다.

```
int factorial(int n) {
    if(n == 0)   return 1;
    else         return n*factorial(n-1);
}
```

이를 template 을 이용하여 아래와 같이 바꿀 수 있다. 이렇게 template 을 이용하는 방법의 장점은 실제 factorial 계산을 런타임에 하는 대신 컴파일 타임에 할 수 있어서 프로그램의 실행시간을 단축시킬 수 있다는 장점이 있다. 이 때 빈칸에 필요한 코드를 template specialization 을 이용하여 작성하시오. (8 점)

```
template<int N>
struct Factorial {
    enum { value = N * Factorial<N-1>::value };
};

```

```
int main()
{
    using namespace std;

    int a = Factorial<5>::value;
    cout << a << endl;
    return 0;
}
```

7. 아래의 코드는 C++11 표준 라이브러리에서 제공하는 스마트 포인터 중 하나인 unique_ptr 을 간략화하여 구현한 클래스이다. 아래의 코드를 완성하기 위한 move constructor 와 move assignment operator 의 코드를 작성하시오. (8 점)

```
template<class T> class my_unique_ptr {
public:
    my_unique_ptr() : m_ptr(nullptr) {}
    explicit my_unique_ptr(T* p) : m_ptr(p) {}
    my_unique_ptr(const my_unique_ptr& p) = delete;
    // move constructor


    ~my_unique_ptr() { delete m_ptr; }

    my_unique_ptr& operator=(const my_unique_ptr& p) = delete;
    // move assignment operator


    T& operator*() { return *m_ptr; }
    const T& operator*() const { return *m_ptr; }

    T& operator->() { return *m_ptr; }
    const T& operator->() const { return *m_ptr; }

private:
    T* m_ptr;
};
```

8. 다음 코드에는 런타임에 발생할 수 있는 한가지 문제점이 존재한다. 이 문제점이 무엇인가? 그리고 이 문제점을 해결하기 위해서는 코드를 어떻게 수정해야 하는가? SomeClass 와 SomeException 클래스는 모두 적절히 정의되어 있다고 가정한다. (8 점)

```
void foo() {
    SomeClass a(1);
    SomeClass* p = new SomeClass(2);
    // ...
    if(b_error) {
        delete p;
        throw SomeException();
    }
    // ...
    delete p;
}

void bar() {
    SomeClass a(3);
    SomeClass* p = new SomeClass(4);
    // ...
    foo();
    // ...
    delete p;
}

int main() {
    try {
        bar();
    }
    catch(SomeException& e) {
        std::cout << e.what() << std::endl;
    }
    return 0;
}
```

9. 다음 코드에서 C-style 의 type cast 연산자들을 각각 그에 대응하는 static_cast, const_cast, reinterpret_cast, dynamic_cast 로 변경한 코드를 작성하시오. (8 점)

```
class Base {
    // ...
};

class Derived : public Base {
    // ...
};

int main() {
    int a = 5;
    double b;
    const int* c = &a;

    b = (double)a;
    int* d = (int*)c;
    int f = 0xA0000000;
    unsigned char* e = (unsigned char*)f;
    Base* p = new Derived;
    Derived* p2 = (Derived*)p;

    return 0;
}
```

10. 다음 코드의 Vec2D 는 2 차원 벡터를 다루기 위한 template class 이다. 아래 코드의 main 함수에서는 Vec2D<int>와 Vec2D<float>의 서로 다른 데이터 타입의 Vec2D 오브젝트 사이의 덧셈 연산을 수행하는 것을 보여준다. 이 덧셈 연산을 구현하기 위한 operator+ 의 오버로딩이 파란색으로 표시되어 있다. 여기서 이 코드에 필요한 적절한 prototype 을 빈칸에 추가하시오. 오버로딩된 operator+는 template function 으로 되어 있으며, Vec2D<int>와 Vec2D<float>, 또는 Vec2D<double>과 Vec2D<long>과 같은 임의의 서로 다른 데이터 타입의 Vec2D 오브젝트 간의 덧셈 연산을 지원한다. 또한 Vec2D<int> 오브젝트와 Vec2D<float> 오브젝트의 덧셈 연산 시 결과로 나와야 되는 타입 (Vec2D<float>) 이 자동으로 결정된다. (4 점)

```
template<typename T>
class Vec2D {
public:
    Vec2D() {}
    Vec2D(T x_, T y_) : x(x_), y(y_) {}
    T x, y;
};

```

```
{
    return Vec2D<decltype(a.x+b.x)>(a.x + b.x, a.y + b.y) ;
}

template<typename T>
std::ostream& operator<<(std::ostream& os, const Vec2D<T>& v)
{
    os << "[" << v.x << ", " << v.y << "]";
    return os;
}

int main()
{
    using namespace std;
    Vec2D<int> a(1,2);
    Vec2D<float> b(3.0f, 2.0f);
    cout << a << " + " << b << " = " << a+b << endl;
    return 0;
}
```

11. 다음 코드의 지역 변수인 data 는 integer 값들을 담고 있는 vector 이다. 아래 코드에서 파란색으로 표시된 코드는 이 data 의 각각의 값들의 조사하여 upper_bound 보다 큰 값들은 upper_bound 로 값을 변경하고, lower_bound 보다 작은 값들은 lower_bound 로 변경하는 작업을 수행한다. 아래 파란색으로 표시된 코드와 동일한 작업을 수행하는 3 가지의 코드를 작성하시오. 각각의 코드는 아래에 A,B,C 로 표시된 내용을 각각 활용하여 작성되어야 한다. (12 점)

    A. for-loop 와 iterator 이용

    B. std::transform 함수와 lambda 함수 이용

    C. std::for_each 함수와 lambda 함수 이용

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>

int main()
{
    using namespace std;

    vector<int> data{11, 17, 4, 10, 29, 4, 18, 18};

    int upper_bound = 20;
    int lower_bound = 10;

    for(int i = 0; i < data.size(); i++) {
        if(data[i] > upper_bound) {
            data[i] = upper_bound;
        }
        else if(data[i] < lower_bound) {
            data[i] = lower_bound;
        }
    }

    for(auto x: data) cout << x << " ";
    cout << endl;

    return 0;
}
```