C++ Introduction Middle Test Summary

1. **C언어에서 문법 계승**

- C언어에서 문법 계승 - C의 절차적 문법과 저수준 접근(포인터 등) 대부분 계승 - 기존 C 코드와 높은 호환성 유지

```
// C 스타일 입출력과 포인터 활용
#include <cstdio>

int main() {
    int value = 42;
    int* ptr = &value;            // 포인터로 변수 주소 저장
    std::printf("Value: %d\n", *ptr);  // 포인터 역참조하여 값 출력
    return 0;
}
```

2. **C++ 설계 목적** -> 객체지향 도입
   -> 타입 체크
   -> 효율성 저하 최소화
   -> Function Overloading : Declare Same name but other Function
   -> Default Parameter : set parameter's default value (ex. void function(int a, int b = 10))
   -> Reference : using Alias, do reference for variable. Alias's reference can't change
   -> Call-By-Reference
   -> new/delete operation
   -> operation re-definition
   -> generic function and class : generic function is the template for function(it can use for other data type allocating)

```cpp
// 함수 오버로딩, 기본 인자, 참조 전달 예시
#include <iostream>
#include <iomanip>

void func(int x) {
    std::cout << "Func int: " << x << "\n";
}
void func(double x, int precision = 2) {
    std::cout << "Func double: " << std::fixed << std::setprecision(precision) <<
x << "\n";
}

void refFunc(int& x) {
    x += 5;
}

int main() {
    func(10);
    func(3.14159);
    int a = 1;
    refFunc(a);
    std::cout << "a after refFunc: " << a << "\n";
    int* p = new int(20);
    std::cout << "dynamic: " << *p << "\n";
    delete p;
    return 0;
}
```

3. **C++ 객체 지향 특성** -> Encapsulation : make shell for preventing object data and security
   -> Object and Instance : Object is the structure make instance
   -> Inheritance : Between class inherit members
   -> Polymorphsim : Using One operand works many operation

```cpp
// 캡슐화, 상속, 다형성 예시
#include <iostream>

class Base {
protected:
    int value;
public:
    Base(int v): value(v) {}
    virtual void show() const { std::cout << "Base value: " << value << "\n"; }
    virtual ~Base() {}
};

class Derived : public Base {
public:
    Derived(int v): Base(v) {}
    void show() const override { std::cout << "Derived value: " << value << "\n";
}
};

int main() {
    Base* b = new Derived(99);
    b->show();  // 다형성(동적 다형성)
    delete b;
    return 0;
}
```

4. **Generic Programming** -> Generic function : the function make same code adopt many data type
   -> Generic class : the class make same code adopt many data type

```cpp
// 템플릿 함수와 클래스 예시
#include <iostream>

template<typename T>
T genericAdd(T a, T b) {
    return a + b;
}

template<typename T>
class GenericPair {
public:
    T first, second;
    GenericPair(T a, T b): first(a), second(b) {}
    T sum() const { return first + second; }
};

int main() {
    std::cout << genericAdd<int>(5, 6) << "\n";
    GenericPair<double> gp(2.5, 3.5);
    std::cout << "GenericPair sum: " << gp.sum() << "\n";
    return 0;
}
```

5. **C++의 단점** -> C 언어와 호환성을 추구
   -> Good Point
   -> Usage of Legacy Code
   -> Bad Point
   -> Capsulation discipline is breaked (Usage of Global variable & Global function)
   -> To Improve this, use static
   -> if declare static int function, it can use for global variable and function
   -> when use static function, it can only use static member variable
   -> Can't use this pointer

6. **네임스페이스(Namespace)** -> 식별자 충돌 방지용 논리적 구획
   -> `namespace` 안에 함수·변수·클래스를 정의하고, 사용 시 네임스코프 지정

```cpp
#include <iostream>

namespace MyMath {
    int add(int a, int b) {
        return a + b;
    }
    int multiply(int a, int b) {
        return a * b;
    }
}

int main() {
    std::cout << "3 + 4 = " << MyMath::add(3, 4) << "\n";
    std::cout << "3 * 4 = " << MyMath::multiply(3, 4) << "\n";
    return 0;
}
```