# Object-Oriented Programming Lab #6

**Department: 화학공학과 Student ID:2019101074 Name:안용상**

## 1. 클래스 인스턴스를 아래와 같이 두가지(a와 p)로 선언하는 경우의 차이점에 대해서 설명하라.

```
class A { /* …  */ };
A a;
A *p = new A;
```

| A a; 선언방식 | A *p = new A;선언방식 |
|---|---|
| 클래스 A의 인스턴스인 a를 생성 | 클래스 A의 인스턴스를 가리키는 포인터 p를 생성 |
| 인스턴스를 스택 메모리에 할당 | 동적으로 인스턴스를 힙 메모리에 할당 |
| a는 선언된 스코프 내에서만 유효 | 인스턴스는 명시적으로 삭제되기 전까지 유지 |
| 해당 스코프를 벗어나면 자동으로 메모리에서 해제 | delete 연산자를 사용하여 객체를 수동으로 삭제 |

## 2. 아래 코드에서 Vector2D는 2차원 벡터로 x, y 요소를 멤버로 가지는 클래스이다. 아래의 코드가 동작하도록 필요한 함수(메소드)를 추가하라.

```
#include <iostream>
class Vector2D {
int x, y;
public:
Vector2D(int x, int y) : x(x), y(y) {}
};

int main() {
Vector2D v1(10, 2), v2(20, 5);
std::cout << v1+v2 << std::endl; // 30, 7 출력
}
```

---

소스코드 :

```
#include <iostream>

class Vector2D {
    int x, y;

public:
    Vector2D(int x, int y) : x(x), y(y) {} // x와 y를 받아 저장하는 생성자

    // 벡터의 덧셈 연산을 정의하기 위해 + 연산자를 오버로딩
    Vector2D operator+(const Vector2D& other) const {
        // newX에 현재 인스턴스의 x멤버와 인자로받은 other인스턴스의 x멤버의 합을 할당
        int newX = x + other.x;
        // newX에 현재 인스턴스의 y멤버와 인자로받은 other인스턴스의 y멤버의 합을 할당
        int newY = y + other.y;
        return Vector2D(newX, newY); // 그리고 그 newX와 newY를 생성자에 전달해서
// 만들어진 Vector2D클래스의 인스턴스를 반환
    }

    // x, y 값을 출력하기 위한 << 연산자를 오버로딩
```

```
    friend std::ostream& operator<<(std::ostream& os, const Vector2D& vector) {
  // 인자로 받은 vector인스턴스의 x멤버를 os에 담고 ", "를 담고 y를 os에 담은 뒤
        os << vector.x << ", " << vector.y;
// 그 os를 반환한다.
        return os;
    }
};

int main() {
    Vector2D v1(10, 2), v2(20, 5);//각각 인자로 전달한 x와 y를 이용해 v1,v2인스턴스를 생성
    std::cout << v1 + v2 << std::endl; //미리 구현한 Vector2D클래스간의 +연산을 수행해서
// 각 인스턴스간의 x와 y의 합을 낸 뒤, 그 합해진 x와 y를 x와 y로 가지는 새로운 인스턴스를 반환.
// 그 반환값을 받아, 미리 구현한 <<연산자로 , 그 반환인스턴스의 x와 y를 x , y형태로 출력
// 30, 7 출력
    return 0;
}
```

결과

```
30, 7

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 13972개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

# 3. 아래 코드가 동작하도록 클래스 A를 완성하라.

```
#include <iostream>
class A {
  int x;
public:
  A(int x) : x(x) {}
  // GetX, Max 함수 작성
};
int main() {
  const A a1(10);
  A a2(5), a3(3);
  std::cout << a1.GetX() << std::endl; // 10 출력
  std::cout << a2.GetX() << std::endl; // 5 출력
  A* p = a2.Max(&a3); // Max: a2.x와 a3.x를 비교하여
  // 큰 값을 가지는 인스턴스 주소 반환
  std::cout << p->GetX() << std::endl; // 5출력
}
```

소스코드

```
#include <iostream>
class A {
  int x;
public:
  A(int x) : x(x) {}
  // GetX, Max 함수 작성
  int GetX() const {
    // private멤버 x를 반환하는 코드를 작성했다
  // 이 경우, 대상 인스턴스가 const A이기 때문에, 이 함수도 const함수로 만들어준다.
    return x;
  }
  A* Max(A* a) const{
    // A클래스 a 인스턴스의 주소를 받아서, A클래스의 주소타입을
// const로 반환하는 Max함수를 작성한다.

    if (a->x > x) {
// 주소값에서 멤버로 접근 할 수 있는 ->연산을 통해 a라는 A클래스 인스턴스를 가리키는
// 주소의 실제 A인스턴스의 x멤버를 가지고 온 뒤, 현재 인스턴스의 x보다 큰지 비교
      return const_cast<A*>(a);
// 만약 크다면 일반 A클래스의 주소인 a를 const A클래스의 주소로 const_cast<A*>해준 뒤에
// 반환
    }
```

```
    else {
// 만약 작다면 일반 A클래스의 주소인 현재인스턴스의 주소this를
// const A클래스의 주소로 const_cast<A*>해준 뒤에
// 반환
      return const_cast<A*>(this);
    }
  }
};
int main() {
  const A a1(10); // a1.x를 10으로 할당 하면서 A클래스 a1생성
  A a2(5), a3(3); // a2와 a3도 각각 a2.x와 a3.x를 5와 3으로 할당하며 생성
  std::cout << a1.GetX() << std::endl; // 10 출력
// GetX() 메서드를 통해 const A타입의 private한 x멤버를 const GetX함수를 통해 얻음
  std::cout << a2.GetX() << std::endl; // 5 출력
// GetX() 메서드를 통해 const A타입의 private한 x멤버를 const GetX함수를 통해 얻음
  A* p = a2.Max(&a3);
// Max: a2.x와 a3.x를 비교하여
// 큰 값을 가지는 인스턴스 주소 반환
  std::cout << p->GetX() << std::endl; // 5출력
}
```

결과

```
10
5
5

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 13616개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

# 4. 클래스의 static 멤버의 특징과 사용법에 대해 설명하라.

☐ 특징

| static멤버의 특징 | 상세설명 |
|---|---|
| 01 . 클래스 수준의 공유 | static 멤버는 클래스 자체에 속하므로, 해당 클래스의 모든 인스턴스에서 공유 즉, 여러 개의 객체가 생성되어도 static 멤버는 하나의 복사본만 존재 따라서 불필요한 멤버생성을 자제하여 메모리를 절약하고 객체 간의 데이터 공유를 가능하게 한다. |
| 02 . 객체에 속하는 멤버가 아님. | static 멤버는 특정 객체에 속하는 멤버가 아니므로 객체를 생성하지 않고도 접근할 수 있다 클래스 이름을 사용하여 접근하며, 객체의 생성과는 독립적으로 동작 |
| 03 . 상수 static 멤버 | static const 형식으로 선언된 멤버는 클래스 수준의 상수로 사용될 수 있다. 이러한 상수 멤버는 클래스의 모든 인스턴스에서 동일한 값을 가지며, 수정할 수 없다. |

☐ 사용법

1. 정적 멤버 변수

```
#include <iostream>
class MyClass {
public:
    static int count; // 정적 멤버 변수 선언

    MyClass() {
        count++; // 객체가 생성될 때마다 count 증가
    }
};
```

```
int MyClass::count = 0; // 정적 멤버 변수 정의 및 초기화

int main() {
    MyClass obj1;// 객체 생성 count 1증가
    MyClass obj2;// 객체 생성 count 1증가
    MyClass obj3;// 객체 생성 count 1증가

    std::cout << "Count: " << MyClass::count << std::endl; // 클래스 이름으로 접근
    return 0;
}
```

### 출력

```
Count: 3

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 6784개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

2. 정적 멤버 메소드

   인스턴스 내부의 멤버와 관련없이 수행되는 동작의 메소드는 static member메소드로 두고 작성해도 된다

```
#include <iostream>
class Utils {
public:
    static int add(int a, int b) {
        return a + b;
    }
};

int main() {
    int result = Utils::add(3, 4); // 클래스 이름으로 접근
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

### 출력

```
Result: 7

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 14916개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

3. 정적 상수 멤버

   클래스 단위로 이용되는 불변의 상수는 인스턴스생성마다 새로 할당할게 아니라, 클래스 단위로 static const int로 지정한다.(메모리 절약)

```
#include <iostream>
class Constants {
public:
    static const int MAX_VALUE = 100;
};

int main() {
    std::cout << "Max value: " << Constants::MAX_VALUE << std::endl;
    return 0;
}
```

### 출력

```
Max value: 100

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 14960개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

# 5. friend 함수와 friend 클래스에 대해 설명하라.

## 특징

- friend 함수와 friend 클래스는 C++에서 접근 제어의 예외를 제공하는 기능이다
- friend를 사용하여 특정 함수나 클래스를 선언함으로써, 해당 함수나 클래스는 접근 제어에 따른 제한 없이 다른 클래스의 private 멤버에 접근할 수 있다
- 이를 통해 두 클래스 간의 친밀한 관계를 형성하거나 특정 함수를 편리하게 사용할 수 있습니다.

## 사용법은 다음과 같다.

1. friend가 될 함수나 클래스의 선언부 가장 앞부분에 friend를 입력한다

```
class MyClass {
private:
    int privateData;

public:
    MyClass(int data) : privateData(data) {}
    friend void FriendFunction(MyClass& obj); // friend 함수 선언을 한다.
// friend를 반환 타입보다 먼저 써준다
};
```

2. 이후 friend함수를 정의한다.

   이때 해당 함수에서 객체의 private한 멤버를 직접 접근하는 코드를 짤 수 있다.

```
void FriendFunction(MyClass& obj) {
    std::cout << "바꾸기전 : " << obj.privateData << std::endl;
    obj.privateData = 42; // friend 함수에서 private 멤버에 접근하는 코드를 짤 수 있다.
    std::cout << "바꾼 후 : " << obj.privateData << std::endl;
}
```

3. 이후 Main함수에서 이를 문제없이 사용한다

```
int main() {
    MyClass obj(10);
    FriendFunction(obj);
    return 0;
}
```

## 전체 소스코드

```
#include <iostream>
class MyClass {
private:
    int privateData;

public:
```

```
    MyClass(int data) : privateData(data) {}
    friend void FriendFunction(MyClass& obj); // friend 함수 선언
};
void FriendFunction(MyClass& obj) {
    std::cout << "바꾸기전 : " << obj.privateData << std::endl;
    obj.privateData = 42; // friend 함수에서 private 멤버에 접근
    std::cout << "바꾼 후 : " << obj.privateData << std::endl;
}
int main() {
    MyClass obj(10);
    FriendFunction(obj);
    return 0;
}
```

### 출력

```
바꾸기전 : 10
바꾼 후 : 42

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 12748개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

# 6. C++에서 class와 struct의 차이점에 대해서 설명하라.

C++에서 `class`와 `struct`는 기본적으로 동일한 기능을 가지는 사용자 정의 데이터 형식이다.

둘 다 멤버 변수와 멤버 함수를 포함할 수 있으며, 객체를 생성하여 사용할 수 있다.

그러나 둘은 차이가 있다.

그 차이는 다음과 같다.

| 차이점 | |
|---|---|
| 1. 기본 접근 제어 | class: 기본적으로 멤버 변수와 멤버 함수의 접근 제어는 private struct: 기본적으로 멤버 변수와 멤버 함수의 접근 제어는 public |
| 2. 상속 | class: 기본적으로 private 상속, protected 상속, public 상속이 모두 가능 struct: 기본적으로 public 상속만 가능 |
| 3. 일관성 | class: 멤버 변수와 멤버 함수가 클래스 내에서 논리적으로 그룹화되어 일관성을 유지하는 것을 강조 struct: 멤버 변수와 멤버 함수가 서로 연관성이 없는 데이터들로 구성될 수 있으며, 일관성에 대한 강제가 덜함 |

# 7. 큰 범위의 0과 양의 정수를 저장하는 BigUnsigned 클래스를 아래와 같은 특징을 가지도록 정의하라. (테스트 코드도 작성해서 결과로 포함)

(a) 정수를 저장하는 멤버는 std::vector로 각 요소는 정수의 각 자릿수로 0-9까지의 정수를 가진다.
(b) 기본 생성자는 정수를 0으로 초기화한다.
(c) unsigned int 형을 파라메터로 하는 생성자를 가지며, 내부 정수를 인수로 초기화한다.
(d) std::string 형을 파라메터로 하는 생성자를 가지며, 내부 정수를 인수로 초기화하며 인수는 n번째 위치의 문자는 내부 정수의 n번째 자릿수이다.
(e) +연산자로 덧셈이 가능하며, std::cout과 <<을 이용하여 정수의 출력이 가능하다.

## 소스코드

```
#include <iostream>
#include <vector>
#include <string>
class BigUnsigned {
    std::vector<int> digits; // 인수로 받은 수의 한자리 한자리수마다 요소로 만들어
                             //push_back할 벡터를 만든다. 이름은 digits
public:
    BigUnsigned() { //디폴트 생성자로써, digits벡터에 0 하나만 요소로써 push_back한다
        digits.push_back(0);
    }

    BigUnsigned(unsigned int num) { // unsigned int 를 인자로 받을 때의 생성자로,
        // 해당 인자 숫자의 낮은 자리수의 숫자부터, 먼저 digits 벡터에 push_back한다.
        while (num > 0) { // while문을 이용해서 가장 낮은 자리수부터 가장 높은 자리수
// 까지 빠짐없이 넣기
            digits.push_back(num % 10);
            num /= 10;
        }
    }

    BigUnsigned(const std::string& str) {
// 문자열로 인자를 받을 때의 생성자이다
        for (int i = str.length() - 1; i >= 0; i--) {
// for문을 통해서, 문자열의 길이를 구하고, 그 길이만큼 for 문을 반복한다
// 반복시행마다, 문자열의 0번째 인덱스부터 마지막 인덱스까지, 해당 숫자문자의 아스키코드
// 빼기 '0'의 아스키코드 값을 구해서, 같은 수를 인트형으로 재해석해서 digits벡터에
// push_back한다.
            digits.push_back(str[i] - '0');
        }
    }

    BigUnsigned operator+(const BigUnsigned& other) const {
// +연산자를 오버로딩한다.
        BigUnsigned result;
// BigUnsigned클래스의 result인스턴스를 디폴트로 생성한다.
        int carry = 0;
// 각 자리수의 합이 10을 넘을때, 그 십의 자리수를 담을 carry변수를 0으로 초기화
        int maxSize = std::max(digits.size(), other.digits.size());
// 더하기 연산을 할 두 BigUnsigned클래스 인스턴스의 digits 벡터멤버의 크기를 비교해서
// 큰것의 size를 maxSize에 담는다
        for (int i = 0; i < maxSize; i++) {
// maxSize, 즉 두 인스턴스의 각각의 벡터를 모두 순회하기 위해,
// 벡터크기가 보다 큰 인스턴스의 벡터의 사이즈만큼 for문을 돌린다.
            int sum = carry;
//매 for시행마다 각 자릿수의 합을 담을 sum변수를 carry로 새로 초기화한다.
// carry로 초기화하는 이유는, 이전 자리수 계산이 끝나고 다음 자리수 계산으로 넘어갈 때
// 10의자리수를 초과한 경우에, 그 10자리 정보가 carry에 반영되어있고
// 다음자리수 계산때, 그 정보를 사용해, 덧셈 반영해줘야하기 때문이다.
            if (i < digits.size()) {
// +오퍼레이터 앞에 있는 인스턴스의 digits의 사이즈가 for문에 의한 i인덱스보다 큰경우
// 즉, 해당 인스턴스의 벡터 사이즈 크기만큼 아직 for문 순회가 덜되었을 때,
// digits의 i번째 인덱스를 sum에 더해준다.
                sum += digits[i];
            }

            if (i < other.digits.size()) {
// +오퍼레이터 뒤에 있는 other인스턴스의 digits의 사이즈가 for문에 의한 i인덱스보다 클때
// 즉, 해당 인스턴스의 벡터 사이즈 크기만큼 아직 for문 순회가 덜되었을 때,
// others.digits의 i번째 인덱스를 sum에 더해준다.
                sum += other.digits[i];
            }
            if (i == 0) {
//두 인스턴스의 i번째 인덱스의 자리수 정수를 모두 더해주고 이전 자리수 계산에서
//있었을 지 모를 10의 자리 수인 carry변수를 덧셈반영한 sum을 이젠
// 10으로 나눠 그 나머지는, 새로 생성한 result의 i번째 인덱스에 push_back해준다.

// 다만 이때 result가 처음에 디폴트 생성자로 초기화되었으므로, 0번째 인덱스가 이미
// 0 으로 초기화 되었으므로, 만약에 i 가 0일때는, result_digits의 [0]인덱스의
//값을 push_back이 아닌 sum%10으로 대체해주는 할당연산을 해준다
                result.digits[0] = sum % 10;
            }
            else {
                result.digits.push_back(sum % 10);
            }
```

```
            carry = sum / 10;
        }
//for 문이 모두 끝난 후 , 마지막 자리수의 상호 계산에서 이 덧셈계산이 10의 자리수를 만들
// 때 이 경우도 끝까지, carry에 반영해준 뒤, result.digits에 push_back해준다.
        if (carry > 0) {
            result.digits.push_back(carry);
        }
        return result;
    }
    friend std::ostream& operator<<(std::ostream& os, const BigUnsigned& num) {
//출력을 해주는 friend함수를 만든다. operator <<에 오버로딩을 한다.
        for (int i = num.digits.size() - 1; i >= 0; i--) {
// 인자로 전달받은 BigUnsigned클래스의 num인스턴스의 사이즈만큼 for문을 돌리되
// num인스턴스 사이즈-1부터 0까지 i가 순차적으로 낮아지는 for문 순회를 한다.
            os << num.digits[i];
//왜냐하면, digits벡터에 낮은 자리수부터 push_back을 해줬으니까
//사람이 읽을 수 있게 출력하려면, 높은 자리수부터 먼저 출력이 되어야하므로
//가장 높은 인덱스의 요소부터 추출해 출력을 해준다
// ostream os실체변수에 이를 담아준다.
        }
        return os;
// ostream os를 반환한다.
    }
};

int main() {
    BigUnsigned num1; // 기본 생성자로 초기화 (0)
    BigUnsigned num2(12345); // unsigned int로 초기화
    BigUnsigned num3("987654321"); // std::string으로 초기화

    BigUnsigned sum = num1 +  num2 + num3; // 각각 다르게 초기화 한 인스턴스를
//모두 더해준다. 이는 덧셈이 가능한데, +오퍼레이터를 오버로딩 해서 가능하게
//구현해 놓았기 때문이다.

    std::cout << "Number 1: " << num1 << std::endl; // num1을 <<오퍼레이터 오버로딩한
//연산자를 이용해서, digits벡터의 높은 인덱스부터 한 요소씩 차례로 출력
    std::cout << "Number 2: " << num2 << std::endl;// num2을 <<오퍼레이터 오버로딩한
//연산자를 이용해서, digits벡터의 높은 인덱스부터 한 요소씩 차례로 출력
    std::cout << "Number 3: " << num3 << std::endl;// num3을 <<오퍼레이터 오버로딩한
//연산자를 이용해서, digits벡터의 높은 인덱스부터 한 요소씩 차례로 출력
    std::cout << "Sum: " << sum << std::endl;// sum을 <<오퍼레이터 오버로딩한
//연산자를 이용해서, digits벡터의 높은 인덱스부터 한 요소씩 차례로 출력

    return 0;
}
```

## 실행 결과

```
Number 1: 0
Number 2: 12345
Number 3: 987654321
Sum: 987666666

C:\Users\admin\source\repos\cpp\x64\Debug\cpp.exe(프로세스 6872개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```