2018 Autumn 심화프로그래밍 Week #7.

학번: **이름:**

1. 원 (Circle) 을 클래스로 구현하려고 한다. 밑의 사진을 참고하여 조건에 맞게 **원 클래스**와 **compareCircle()** 함수를 구현하고 그 결과를 출력하시오. 단, 파이는 3.0으로 계산하시오. (25 pt)

```
class Circle
{
private:
    int radius;                 // 원의 반지름

public:
    double getArea();           // 원의 면적
    double getCircumference();  // 원의 둘레
```

```
int main()
{
    Circle circle1(4);
    Circle circle2(2);

    compareCircle(circle1, circle2);

    return 0;
}
```

```
반지름이 더 큰 원의 면적은 : 48
반지름이 더 큰 원의 둘레는 : 24
```

조건 1) Circle 클래스는 **원의 반지름**을 멤버로 가지며 이를 통해 **원의 면적과 둘레**를 계산한다.

조건 2) comPareCircle() 함수는 **두 원의 반지름**을 비교하여 **더 큰 원의 면적과 둘레를 출력**한다. 이때 편의상 $\boldsymbol{\pi = 3}$ 으로 계산한다.

2. 스택(Stack) 클래스를 구현하려고 한다. 밑의 사진을 참고하여 Stack 클래스를 구현하고 그 결과를 출력하시오. (25 pt)

```
class Stack
{
private:
    int size;               // 현재 스택의 사이즈
    int maxSize;            // 스택의 최대 사이즈
    int *stack;             // element가 저장될 stack, 배열 동적할당

public:
    Stack(int maxSize);     // 생성자
    ~Stack();               // 소멸자

    void push(int element); // 스택에 element를 삽입
    void pop();             // 스택의 맨 위 element를 출력 후 삭제
    void getSize();         // 현재 스택의 사이즈 반환
};
```

```
int main()
{
    const int MAX_SIZE = 5; // 스택의 최대 사이즈
    Stack stack(MAX_SIZE);

    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.push(5);
    stack.push(6);

    stack.pop();
    stack.pop();
    stack.pop();
    stack.pop();
    stack.pop();
    stack.pop();

    return 0;
}
```

```
maxSize : 5

push : 1
push : 2
push : 3
push : 4
push : 5
Stack is Full!

pop : 5
pop : 4
pop : 3
pop : 2
pop : 1
Stack is Empty!
```

Stack 에 대한 설명은 강의 7.11 "std::vector를 스택 처럼 사용하기" 를 참고하세요.

조건 1) 생성자 에선 **배열 동적 할당**을 통해 스택을 구현하며 소멸자 에서 동적 할당한 배열을 **해제**한다.

조건 2) 스택의 push() 함수는 **스택이 현재 꽉 찬 상태**라면 "Stack is Full!" 을 출력한다.

조건 3) 스택의 pop() 함수는 **스택이 현재 빈 상태**라면 "Stack is Empty!" 를 출력한다.

3. 밑의 사진을 참고하여 **은행**과 **고객** 클래스를 구현하여 **입금, 출금** 이 가능한 프로그램을 작성하고 그 결과를 출력하라. (25 pt)

```
class Bank
{
private:
    int balance;        // 은행 잔고

public:
    Bank(int balance) :balance(balance) {   }
};


class Customer
{
private:
    int cash;

public:
    Customer(int cash) :cash(cash) {    }

    void withDraw(Bank& bank, int money);       // 출금
    void deposit(Bank& bank, int money);        // 입금
};
```

```
Bank bank(2000);              // 은행잔고는 2000 원
Customer customer(1000);      // 보유현금은 1000 원

bank.getInfo();
customer.getInfo();

customer.withDraw(bank, 500);          // 은행에서 500원 출금

bank.getInfo();
customer.getInfo();

customer.deposit(bank, 1000);          // 은행에 1500원 입금

bank.getInfo();
customer.getInfo();
```

```
은행 잔고 : 2000
보유 현금 : 1000

출금 : 500
은행 잔고 : 1500
보유 현금 : 1500

입금 : 1000
은행 잔고 : 2500
보유 현금 : 500
```

다른 클래스의 private 멤버에 접근하기 위해선 friend class 를 사용하세요.

조건 1) 입금, 출금 후 **보유 현금과 은행 잔고를 출력**하라.

조건 2) 입금, 출금 에서 은행의 private 멤버인 balance 에 접근하기 위해 **friend class**를 사용하라.

4. 프로그램을 작성하다 보면, 어떠한 클래스는 생성자가 여러 차례 호출되더라도 **실제로 생성되는 객체는 단 1개** 여야 할 때가 있다. 밑의 사진을 참고하여 단 1개의 객체만 생성하는 Single 클래스를 완성하라. (**싱글톤 패턴**을 이용하시오.) (25 pt)

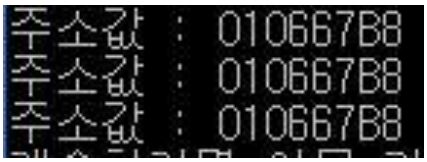
```
class Single
{
private:
    static Single* instance;        // 한 객체만을 생성하기 위한 포인터 변수


    // 생성자를 private 으로 선언하여 외부에서
    // getInstance() 로만 객체 생성이 가능하게 함
    Single() {}
    Single(const Single& other);
    ~Single() {}

public:
    static Single* getInstance() {
        // 구현해야 할 부분...
    }
};

Single* Single::instance = nullptr;
```

```
int main()
{
    Single *a = Single::getInstance();
    Single *b = Single::getInstance();
    Single *c = Single::getInstance();
```



조건 1) getInstance() 가 여러 번 호출되어도 실제로 단 1 개의 객체만을 생성하도록 getInstance() 를 완성하라.

조건 2) 생성한 객체의 주소값을 출력하여 단 1 개의 객체만 생성되었는지를 확인하라.