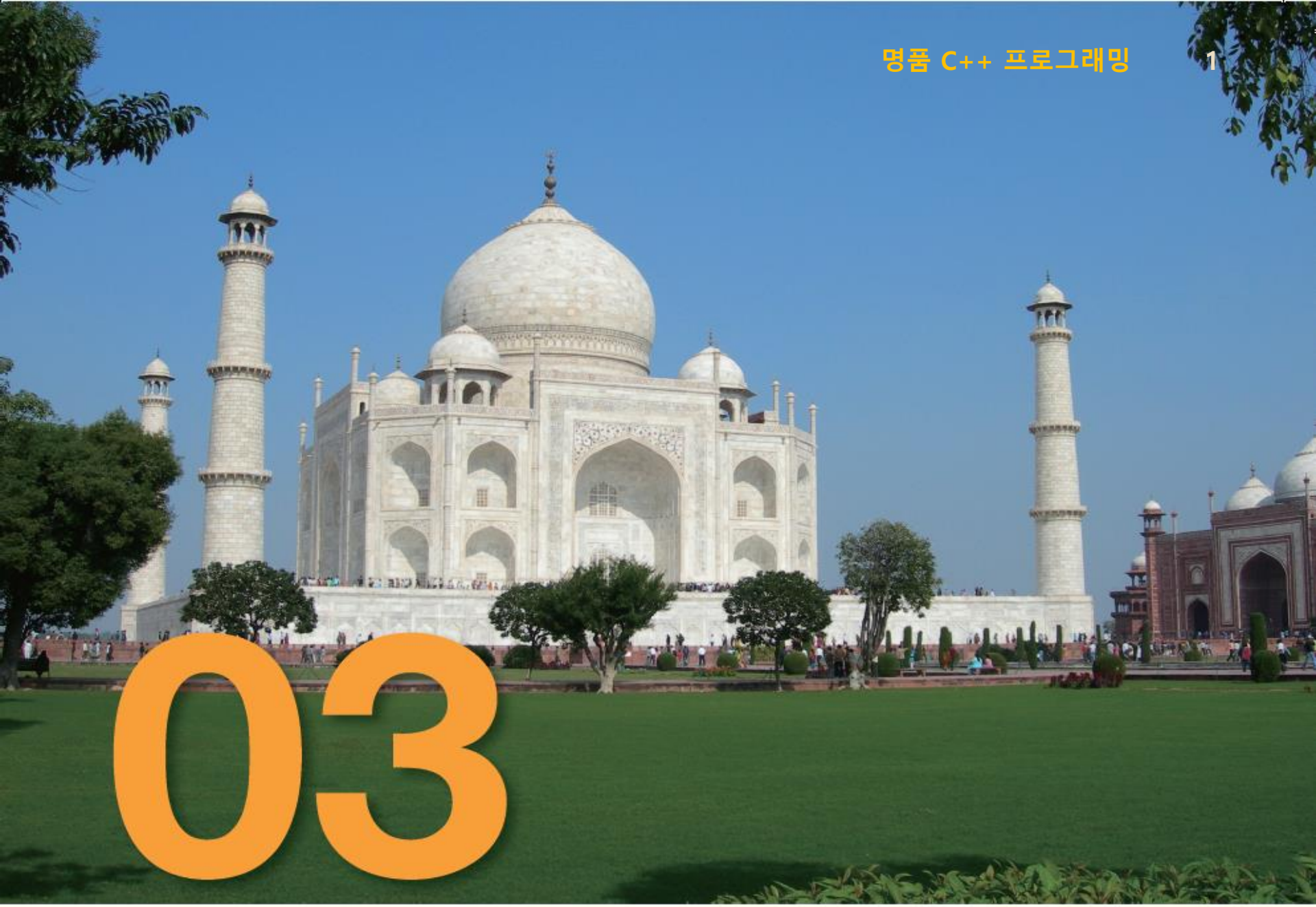
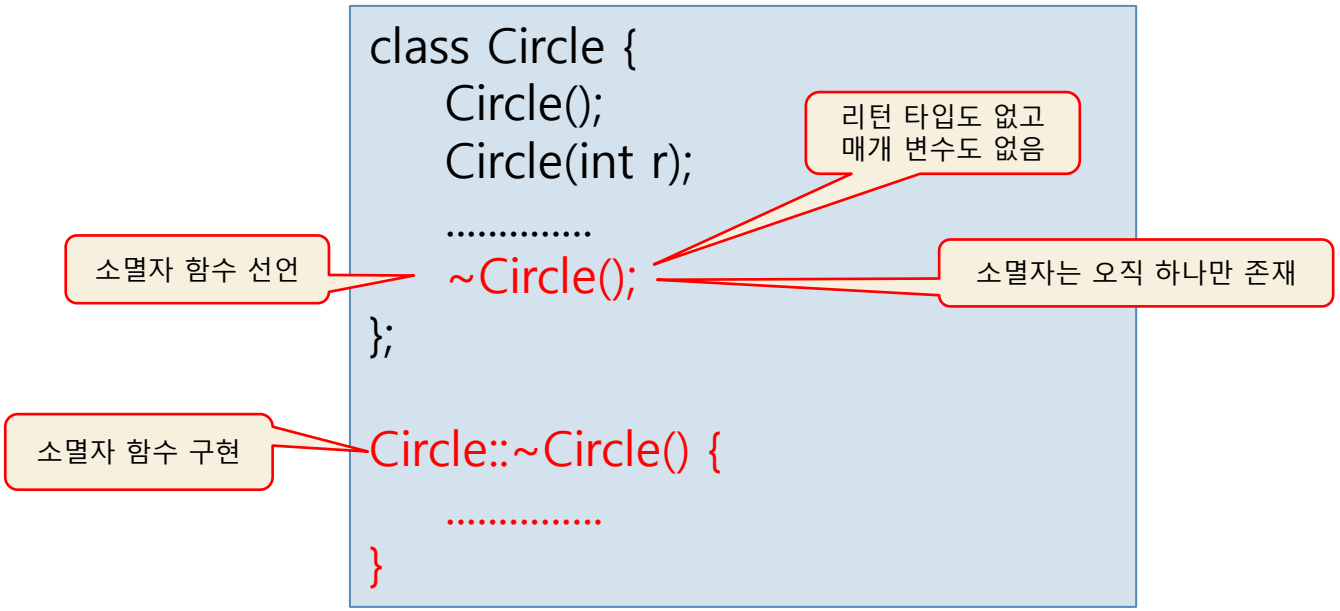


03

클래스와 객체

소멸자

- 소멸자
  - 객체가 소멸되는 시점에서 자동으로 호출되는 함수
    - 오직 한번만 자동 호출, 임의로 호출할 수 없음
    - 객체 메모리 소멸 직전 호출됨



소멸자 특징

- 소멸자의 목적
  - 객체가 사라질 때 마무리 작업을 위함
  - 실행 도중 동적으로 할당 받은 메모리 해제, 파일 저장 및 닫기, 네트워크 닫기 등
- 소멸자 함수의 이름은 클래스 이름 앞에 ~를 붙인다.
  - 예) Circle::~Circle() { ... }
- 소멸자는 리턴 타입이 없고, 어떤 값도 리턴하면 안됨
  - 리턴 타입 선언 불가
- 중복 불가능
  - 소멸자는 한 클래스 내에 오직 한 개만 작성 가능
  - 소멸자는 매개 변수 없는 함수
- 소멸자가 선언되어 있지 않으면 기본 소멸자가 자동 생성
  - 컴파일러에 의해 기본 소멸자 코드 생성
  - 컴파일러가 생성한 기본 소멸자 : 아무 것도 하지 않고 단순 리턴

예제 3-7 Circle 클래스에 소멸자 작성 및 실행


```
#include <iostream>
using namespace std;

class Circle {
public:
    int radius;

    Circle();
    Circle(int r);
    ~Circle(); // 소멸자
    double getArea();
};


Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::~Circle() {
    cout << "반지름 " << radius << " 원 소멸" << endl;
}
```

```
double Circle::getArea() {
   return 3.14*radius*radius;
}

int main() {
    Circle donut;
    Circle pizza(30);

    return 0;
}
```

main() 함수가 종료하면 main() 함수의 스택에 생성된 pizza, donut 객체가 소멸된다.

```
반지름 1 원 생성
반지름 30 원 생성
반지름 30 원 소멸
반지름 1 원 소멸
```

객체는 생성의 반대 순으로 소멸된다.

생성자/소멸자 실행 순서

- 객체가 선언된 위치에 따른 분류
  - 지역 객체
    - 함수 내에 선언된 객체로서, 함수가 종료하면 소멸된다.
  - 전역 객체
    - 함수의 바깥에 선언된 객체로서, 프로그램이 종료할 때 소멸된다.
- 객체 생성 순서
  - 전역 객체는 프로그램에 선언된 순서로 생성
  - 지역 객체는 함수가 호출되는 순간에 순서대로 생성
- 객체 소멸 순서
  - 함수가 종료하면, 지역 객체가 생성된 순서의 역순으로 소멸
  - 프로그램이 종료하면, 전역 객체가 생성된 순서의 역순으로 소멸
- new를 이용하여 동적으로 생성된 객체의 경우
  - new를 실행하는 순간 객체 생성
  - delete 연산자를 실행할 때 객체 소멸

예제 3-8 지역 객체와 전역 객체의 생성 및 소멸 순서


```
#include <iostream>
using namespace std;

class Circle {
public:
    int radius;
    Circle();
    Circle(int r);
    ~Circle();
    double getArea();
};

Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::~Circle() {
    cout << "반지름 " << radius << " 원 소멸" << endl;
}

double Circle::getArea() {
    return 3.14*radius*radius;
}
```

다음 프로그램의 실행 결과는 무엇인가?

```
Circle globalDonut(1000);
Circle globalPizza(2000);

void f() {
    Circle fDonut(100);
    Circle fPizza(200);
}

int main() {
    Circle mainDonut;
    Circle mainPizza(30);
    f();
}
```

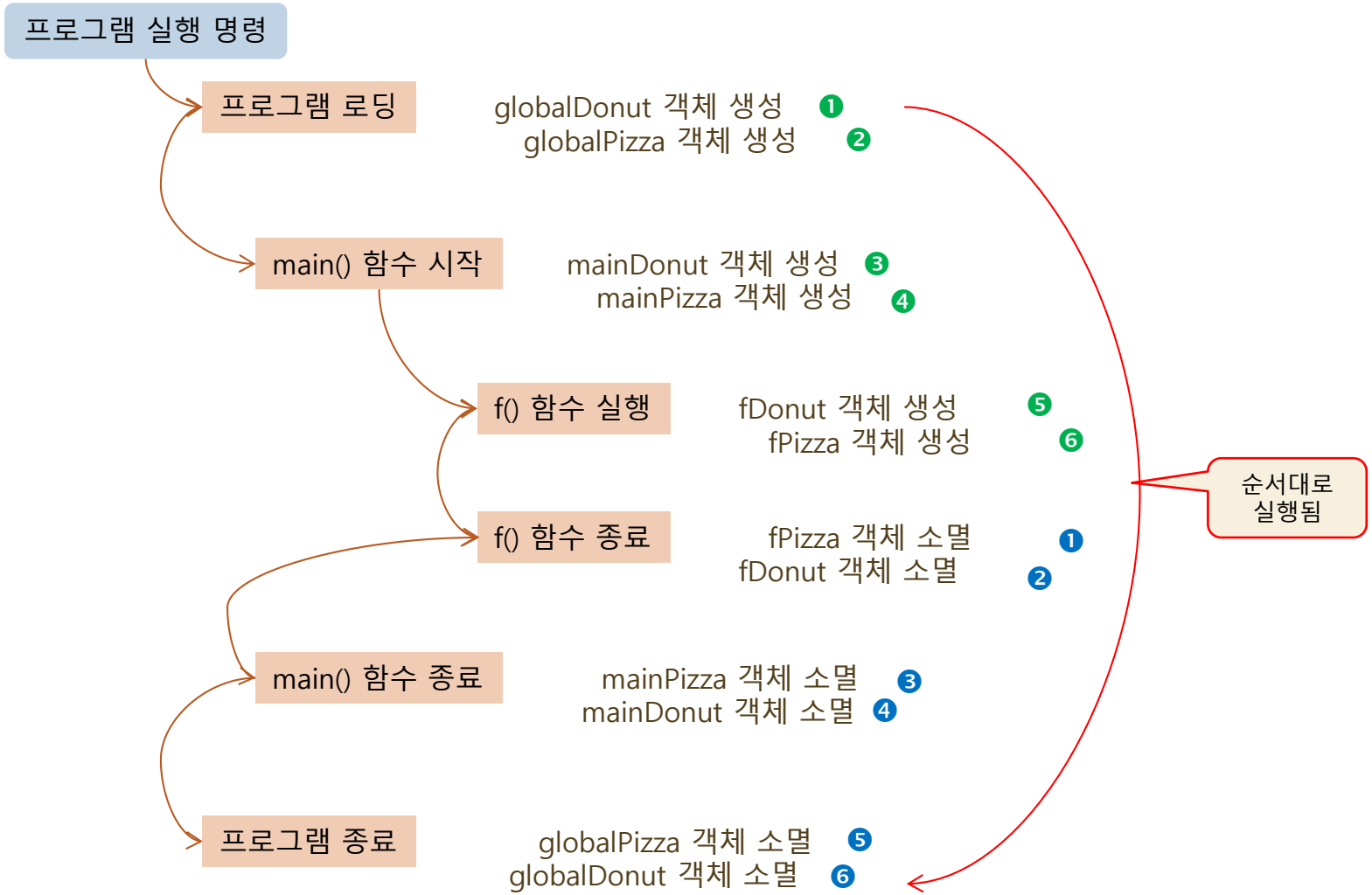
전역 객체 생성

지역 객체 생성

지역 객체 생성

```
반지름 1000 원 생성
반지름 2000 원 생성
반지름 1 원 생성
반지름 30 원 생성
반지름 100 원 생성
반지름 200 원 생성
반지름 200 원 소멸
반지름 100 원 소멸
반지름 30 원 소멸
반지름 1 원 소멸
반지름 2000 원 소멸
반지름 1000 원 소멸
```

예제 3-8의 지역 객체와 전역 객체의 생성과 소멸 과정



접근 지정자

- 캡슐화의 목적
  - 객체 보호, 보안
  - C++에서 객체의 캡슐화 전략
    - 객체의 상태를 나타내는 데이터 멤버(멤버 변수)에 대한 보호
    - 중요한 멤버는 다른 클래스나 객체에서 접근할 수 없도록 보호
    - 외부와의 인터페이스를 위해서 일부 멤버는 외부에 접근 허용

- 멤버에 대한 3 가지 접근 지정자
  - private
    - 동일한 클래스의 멤버 함수에만 제한함
  - public
    - 모든 다른 클래스에 허용
  - protected
    - 클래스 자신과 상속받은 자식 클래스에만 허용

```
class Sample {
private:
   // private 멤버 선언
public:
   // public 멤버 선언
protected:
   // protected 멤버 선언
};
```

중복 접근 지정과 디폴트 접근 지정

접근 지정의 중복 사용 가능

```
class Sample {
private:
   // private 멤버 선언
public:
   // public 멤버 선언
private:
   // private 멤버 선언
};
```

접근 지정의 중복 사례

```
class Sample {
private:
   int x, y;
public:
   Sample();
private:
   bool checkXY();
};
```

디폴트 접근 지정은 private

디폴트 접근 지정은 private

```
class Circle {
   int radius;
public:
   Circle();
   Circle(int r);
   double getArea();
};
```

=

```
class Circle {
private:
   int radius;
public:
   Circle();
   Circle(int r);
   double getArea();
};
```

멤버 변수는 private 지정이 바람직함


```
class Circle {
public:
    int radius;
    Circle();
    Circle(int r);
    double getArea();
};


Circle::Circle() {
    radius = 1;
}
Circle::Circle(int r) {
    radius = r;
}
```

멤버 변수 보호받지 못함

```
int main() {
    Circle waffle;
    waffle.radius = 5;
}
```

노출된 멤버는 마음대로 접근. 나쁜 사례

(a) 멤버 변수를 public으로 선언한 나쁜 사례

```
class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    double getArea();
};

Circle::Circle() {
    radius = 1;
}
Circle::Circle(int r) {
    radius = r;
}
```

멤버 변수 보호받고 있음

```
int main() {
    Circle waffle(5); // 생성자에서 radius 설정
    waffle.radius = 5; // private 멤버 접근 불가
}
```

(b) 멤버 변수를 private으로 선언한 바람직한 사례

예제 3–9 다음 소스의 컴파일 오류가 발생하는 곳은 어디인가?

```
#include <iostream>
using namespace std;

class PrivateAccessError {
private:
    int a;
    void f();
    PrivateAccessError();
public:
    int b;
    PrivateAccessError(int x);
    void g();
};

PrivateAccessError::PrivateAccessError() {
    a = 1;          // (1)
    b = 1;          // (2)
}

PrivateAccessError::PrivateAccessError(int x) {
    a = x;          // (3)
    b = x;          // (4)
}

void PrivateAccessError::f() {
    a = 5;          // (5)
    b = 5;          // (6)
}

void PrivateAccessError::g() {
    a = 6;          // (7)
    b = 6;          // (8)
}
```

```
int main() {
    PrivateAccessError objA;          // (9)
    PrivateAccessError objB(100);     // (10)
    objB.a = 10;                      // (11)
    objB.b = 20;                      // (12)
    objB.f();                         // (13)
    objB.g();                         // (14)
}
```

정답

(9) 생성자 PrivateAccessError()는 private 이므로 main()에서 호출할 수 없다.
(11) a는 PrivateAccessError 클래스의 private 멤버이므로 main()에서 접근할 수 없다.
(13) f()는 PrivateAccessError 클래스의 private 멤버이므로 main()에서 호출할 수 없다.

- 생성자도 private으로 선언할 수 있다. 생성자를 private으로 선언하는 경우는 한 클래스에서 오직 하나의 객체만 생성할 수 있도록 하기 위한 것으로 부록 D의 singleton 패턴을 참조하라.

C++ 구조체

- C++ 구조체
  - 상속, 멤버, 접근 지정 등 모든 것이 클래스와 동일
  - 클래스와 유일하게 다른 점
    - 구조체의 디폴트 접근 지정 – public
    - 클래스의 디폴트 접근 지정 – private
- C++에서 구조체를 수용한 이유?
  - C 언어와의 호환성 때문
    - C의 구조체 100% 호환 수용
    - C 소스를 그대로 가져다 쓰기 위해
- 구조체 객체 생성
  - struct 키워드 생략

```
struct StructName {
private:
   // private 멤버 선언
protected:
   // protected 멤버 선언
public:
   // public 멤버 선언
};
```

```
structName stObj;          // (0), C++ 구조체 객체 생성
struct structName stObj;   // (X), C 언어의 구조체 객체 생성
```

구조체와 클래스의 디폴트 접근 지정 비교

구조체에서
디폴트 접근 지정은
public

```
struct Circle {
    Circle();
    Circle(int r);
    double getArea();
private:
    int radius;
};
```

동일

```
class Circle {
    int radius;
public:
    Circle();
    Circle(int r);
    double getArea();
};
```

클래스에서
디폴트 접근 지정은
private

예제 3-10 Circle 클래스를 C++ 구조체를 이용하여 재작성


```
#include <iostream>
using namespace std;

// C++ 구조체 선언
struct StructCircle {
private:
   int radius;
public:
   StructCircle(int r) { radius = r; }  // 구조체의 생성자
   double getArea();
};

double StructCircle::getArea() {
   return 3.14*radius*radius;
}

int main() {
   StructCircle waffle(3);
   cout << "면적은 " << waffle.getArea();
}
```

```
면적은 28.26
```

바람직한 C++ 프로그램 작성법

- 클래스를 헤더 파일과 cpp 파일로 분리하여 작성
  - 클래스마다 분리 저장
  - 클래스 선언 부
    - 헤더 파일(.h)에 저장
  - 클래스 구현 부
    - cpp 파일에 저장
    - 클래스가 선언된 헤더 파일 include
  - main() 등 전역 함수나 변수는 다른 cpp 파일에 분산 저장
    - 필요하면 클래스가 선언된 헤더 파일 include
- 목적
  - 클래스 재사용

예제 3-3의 소스를 헤더 파일과 cpp 파일로 분리하여 작성한 사례

```
class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    double getArea();
};
```

Circle.h

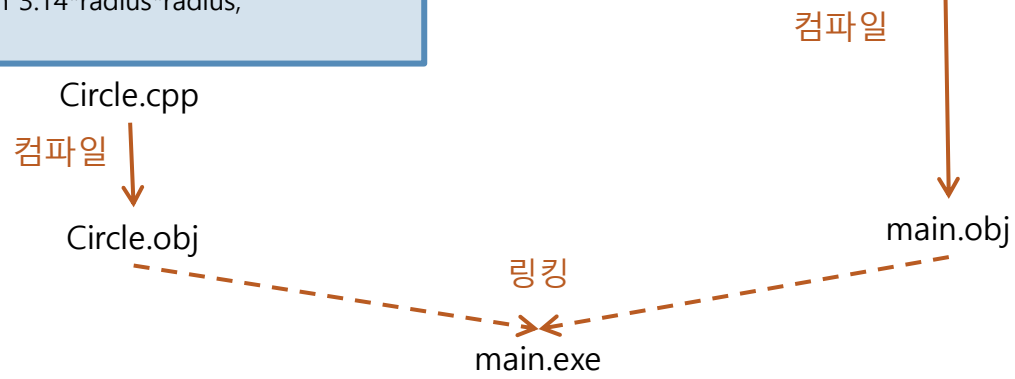
```
#include <iostream>
using namespace std;

#include "Circle.h"

Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius;
    cout << " 원 생성" << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius;
    cout << " 원 생성" << endl;
}

double Circle::getArea() {
    return 3.14*radius*radius;
}
```

Circle.cpp

```
#include <iostream>
using namespace std;

#include "Circle.h"

int main() {
    Circle donut;
    double area = donut.getArea();
    cout << "donut 면적은 ";
    cout << area << endl;

    Circle pizza(30);
    area = pizza.getArea();
    cout << "pizza 면적은 ";
    cout << area << endl;
}
```

main.cpp

```
반지름 1 원 생성
donut 면적은 3.14
반지름 30 원 생성
pizza 면적은 2826
```



헤더 파일의 중복 include 문제

- 헤더 파일을 중복 include 할 때 생기는 문제

```
#include <iostream>
using namespace std;

#include "Circle.h"
#include "Circle.h" // 컴파일 오류 발생
#include "Circle.h"

int main() {
        ..........
}
```

circle.h(4): error C2011: 'Circle' : 'class' 형식 재정의

헤더 파일의 중복 include 문제를 조건 컴파일로 해결

조건 컴파일 문의 상수(CIRCLE_H)는
다른 조건 컴파일 상수와 충돌을 피하기 위해
클래스의 이름으로 하는 것이 좋음.

조건 컴파일 문.
Circle.h를 여러 번
include해도 문제
없게 하기 위함

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    double getArea();
};

#endif
```

Circle.h

```
#include <iostream>
using namespace std;

#include "Circle.h"
#include "Circle.h"
#include "Circle.h"

int main() {
    ..........
}
```

컴파일 오류 없음

main.cpp

include 됨

```
#ifndef CIRCLE_H
#define CIRCLE_H

// Circle 클래스 선언
class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    double getArea();
};

#endif
```

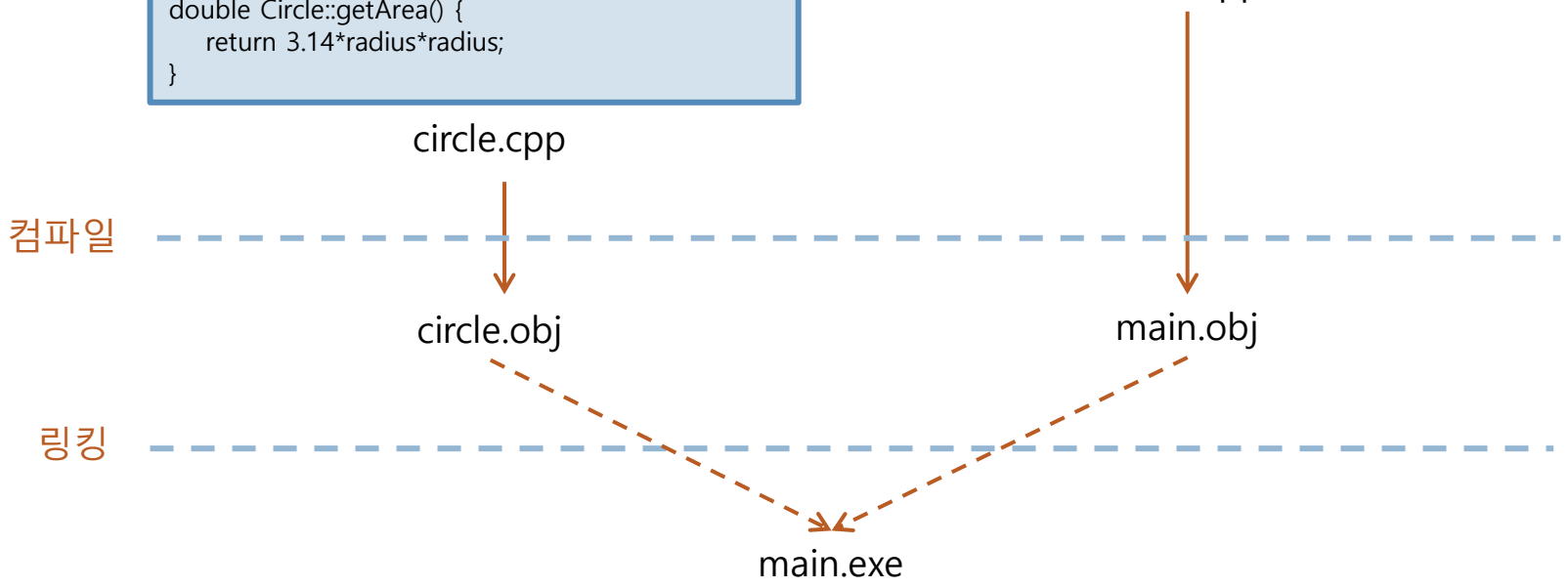
circle.h

```
#include <iostream>
using namespace std;

#include "circle.h"

// CIrcle 클래스 구현. 모든 멤버 함수를 작성한다.
Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius << " 원 생성₩n";
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성₩n";
}

double Circle::getArea() {
    return 3.14*radius*radius;
}
```

circle.cpp

```
#include <iostream>
using namespace std;

#include "circle.h"

int main() {
    Circle donut;
    double area = donut.getArea();
    cout << " donut의 면적은 " << area << "₩n";

    Circle pizza(30);
    area = pizza.getArea();
    cout << "pizza의 면적은 " << area << "₩n";
}
```

main.cpp



예제 3–11 헤더 파일과 cpp 파일로 분리하기

아래의 소스를 헤더 파일과 cpp 파일로 분리하여 재작성하라.

```
#include <iostream>
using namespace std;

class Adder { // 덧셈 모듈 클래스
   int op1, op2;
public:
   Adder(int a, int b);
   int process();
};

Adder::Adder(int a, int b) {
   op1 = a; op2 = b;
}

int Adder::process() {
   return op1 + op2;
}
```

```
class Calculator { // 계산기 클래스
public:
   void run();
};

void Calculator::run() {
   cout << "두 개의 수를 입력하세요>>";
   int a, b;
   cin >> a >> b; // 정수 두 개 입력
   Adder adder(a, b); // 덧셈기 생성
   cout << adder.process(); // 덧셈 계산
}

int main() {
   Calculator calc; // calc 객체 생성
   calc.run(); // 계산기 시작
}
```

```
두 개의 수를 입력하세요>>5 -20
-15
```