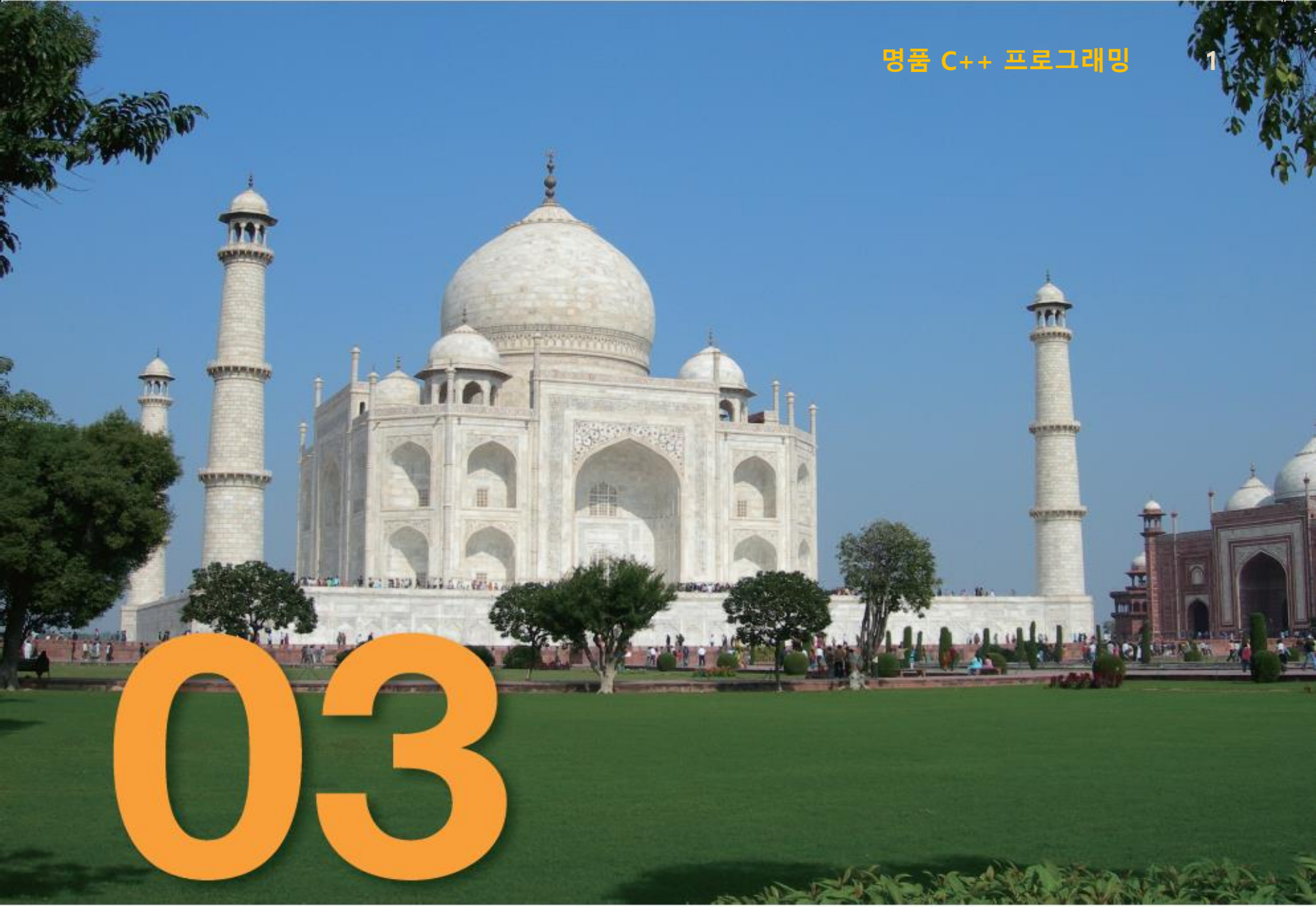


# 03

# 클래스와 객체

# 학습 목표

1. 실세계의 객체와 C++ 객체에 대해 이해한다.
2. C++ 클래스를 작성할 수 있다.
3. 객체를 생성하고 활용할 수 있다.
4. 생성자와 소멸자를 알고 작성할 수 있다.
5. private, protected, public 접근 지정자를 이해한다.
6. 인라인 함수의 목적을 이해하고 활용할 수 있다.

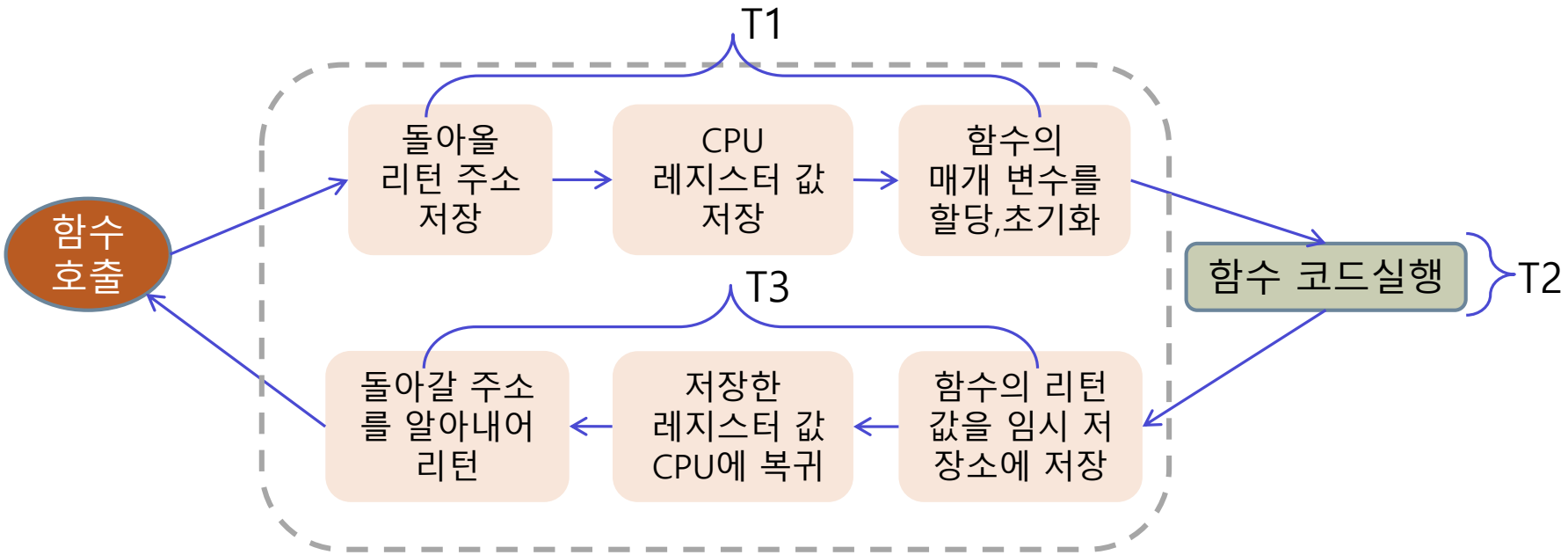
# 함수 호출에 따른 시간 오버헤드



- 함수의 장점
  - 코드의 중복을 방지 -> 코드사이즈를 줄임
  - 코드의 재활용성을 높일 수 있음 -> 개발기간 단축
- 함수의 단점
  - 함수 코드의 실행 전후로 자동으로 추가되는 코드 존재 -> 함수를 사용하지 않는 경우보다 실행시간 길어 짐(오버헤드 발생)
  - 함수 호출이 많아질 경우 또는 함수의 코드가 매우 짧은 경우 함수의 장점이 약해짐

# 함수 호출에 따른 시간 오버헤드






- 함수를 실행하는데 걸리는 총 시간 : T1+T2+T3
- T2 : 함수 몸체에 작성된 코드를 실행하는 데 걸리는 시간
- T1+T3 : 자동으로 추가된 코드를 실행하는 데 걸리는 시간(오버헤드)

# 함수 호출에 따른 시간 오버헤드



```
#include <iostream>
using namespace std;
int odd(int x) {
	return (x % 2);
}
int main() {
	int sum = 0;
	for (int i = 1; i <= 10000; i++)
		if (odd(i)) sum += i;
	cout << sum;
}
```

소스코드

기계어코드

```
; 104  : int odd(int x) {

  00000 55       push    ebp
  00001 8b ec        mov     ebp, esp
  00003 81 ec c0 00 00
    00       sub     esp, 192       ; 000000c0H
  00009 53       push    ebx
  0000a 56       push    esi
  0000b 57       push    edi
  0000c 8b fd        mov     edi, ebp
  0000e 33 c9        xor     ecx, ecx
  00010 b8 cc cc cc cc   mov     eax, -858993460        ; ccccccccH
  00015 f3 ab        rep stosd
  00017 b9 00 00 00 00   mov     ecx, OFFSET __8CC927B7_ch3@cpp
  0001c e8 00 00 00 00   call    @__CheckForDebuggerJustMyCode@4

; 105  :    return (x % 2);

  00021 8b 45 08     mov     eax, DWORD PTR _x$[ebp]
  00024 25 01 00 00 80   and     eax, -2147483647   ; 80000001H
  00029 79 05        jns     SHORT $LN3@odd
  0002b 48       dec     eax
  0002c 83 c8 fe     or  eax, -2            ; fffffffeH
  0002f 40       inc     eax
$LN3@odd:

; 106  : }

  00030 5f       pop     edi
  00031 5e       pop     esi
  00032 5b       pop     ebx
  00033 81 c4 c0 00 00
    00       add     esp, 192       ; 000000c0H
  00039 3b ec        cmp     ebp, esp
  0003b e8 00 00 00 00   call    __RTC_CheckEsp
  00040 8b e5        mov     esp, ebp
  00042 5d       pop     ebp
  00043 c3       ret     0
?odd@@YAHH@Z ENDP                 ; odd
```

T1

T2

T3

# 함수 호출에 따른 시간소모가 심각한 사례



- 함수의 코드 크기가 작은 경우 함수를 사용하여 얻는 이익보다 오버헤드가 너무 큼

```
#include <iostream>
using namespace std;
int odd(int x) {
	return (x % 2);
}
int main() {
	int sum = 0;
	// 1에서 10000까지의 홀수의 합 계산
	for (int i = 1; i <= 10000; i++)
		if (odd(i)) sum += i;
	cout << sum;
}
// 예를 들어 T1+T3=1msec -> 오버헤드 10000*1m=10sec
```

```
25000000
```

# C++ 프로그램의 문제점



- C++에는 멤버함수 호출이 매우 많음
  - 생성자, 소멸자, 멤버 접근 함수 getX, setX를 통하여 멤버변수에 접근하므로 멤버함수 호출이 매우 많아짐
- C++에는 짧은 코드의 멤버 함수가 많음
  - 생성자, 소멸자, setX, getX 함수 등 매우 짧은 코드를 갖는 멤버함수 많음
  - 따라서 함수호출로 인한 오버헤드 증가

# C++ 프로그램의 문제점



- 매우 짧은 코드를 함수로 만들면 코드크기는 거의 비슷하지만 실행시간은 길어짐 -> 장점보다 단점이 큼

```
int func() {
      // 1줄 코드
}
int main() {
      func();
      ...
      func();
      ...
      func();
      ...
      func();
      ...
      func();
}
```

# 인라인 함수



- 인라인 함수
  - 함수 구현부의 머리부분 제일 앞에 inline 키워드로 선언된 함수
  - inline키워드의 위치는 선언부, 구현부 모두 가능하나 구현부에 하는 것을 선호함
- 인라인 함수에 대한 처리
  - 컴파일러에 의하여 인라인 함수를 호출하는 곳에 인라인 함수 코드를 그대로 삽입(소스코드 대입이 아니라 기계어 수준에서 대입)
  - 실행파일에서는 함수 호출이 발생하지 않으므로 실행속도가 빨라짐
- 인라인 함수의 목적
  - C++ 프로그램의 실행 속도 향상
  - 자주 호출되는 짧은 멤버함수 호출에 따른 처리시간 소모를 줄임 -> **객체지향형 언어의 특징(캡슐화, 정보은닉)은 유지하면서 성능 개선**

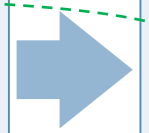
# 인라인 함수 사례



- 컴파일러가 인라인 함수자리에 함수의 코드를 직접 삽입

```
#include <iostream>
using namespace std;
inline int odd(int x) {
	return (x % 2);
}
int main() {
	int sum = 0;
	for (int i = 1; i <= 10000; i++)
	{
		if (odd(i)) sum += i;
	}
	cout << sum;
}
```

소스파일 원본(함수 있음)



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

int main() {
	int sum = 0;

	for (int i = 1; i <= 10000; i++)
	{
		if ((i % 2)) sum += i;
	}
	cout << sum;
}
```

수정된 소스파일(함수 없음)
->실제는 기계어 코드를 대입

# 인라인 멤버 함수 사례



```
class Circle {
private:
	int radius;
public:
	Circle();
	Circle(int r);
	double getArea();
};
inline Circle::Circle() { radius = 1; }			// 인라인 멤버함수
inline Circle::Circle(int r) { radius = r; }		// 인라인 멤버함수
double Circle::getArea() { return 3.14 * radius * radius; }
```

# 인라인 함수 장단점



- 장점
  - 코드가 작은 함수를 인라인으로 선언하면 프로그램의 실행 시간을 단축
- 단점
  - 인라인 함수는 호출하는 곳에 함수 코드를 단순 삽입함으로 함수를 여러 번 호출하게 되면 전체 코드 크기가 증가
  - 짧은 코드의 함수만 인라인으로 선언하는 것이 좋음
- 인라인 제약 사항
  - inline으로 선언한다고 컴파일러가 모두 인라인함수로 처리하지 않음
  - 컴파일러가 판단하여 inline 으로 처리할지를 결정
  - 재귀함수, static 변수, 반복문 등을 가진 함수는 인라인함수로 허용하지 않음

# 자동 인라인 함수



- 자동 인라인 함수
  - 클래스 선언부에 구현된 멤버 함수는 자동으로 인라인 함수로 처리
  - inline 키워드를 사용할 필요 없음
  - 컴파일러에 의해 자동으로 인라인함수 처리
  - 생성자를 포함한 모든 멤버 함수가 자동 인라인 함수 가능

```
class Circle {
private:
	int radius;
public:
	Circle() { radius = 1; }                                  // 자동 인라인함수
	double getArea() { return 3.14*radius*radius; } // 자동 인라인함수
};
```

# 자동 인라인 함수



```
class Circle {
private:
	int radius;
public:
	Circle();
	Circle(int r);
	double getArea();
};
inline Circle::Circle() {
	radius = 1;
}
inline Circle::Circle(int r) {
	radius = r;
}
inline double Circle::getArea() {
	return 3.14*radius*radius;
}
```

(a) 멤버함수를 inline으로 선언하는 경우

=

```
class Circle {
private:
	int radius;
public:
	Circle() { radius = 1;}
	Circle(int r) { radius = r; }
	double getArea() {
		return 3.14*radius*radius;
	}
};
```

(b) 자동 인라인 함수로 처리되는 경우

# C++ 구조체, 바람직한 C++ 프로그램 작성법



- 각자 읽어볼 것

# 실습과제1



- 다음 실행결과가 나오도록 Triangle 클래스를 정의하고 모든 멤버함수를 인라인 함수로 작성하라.

```
// 코드추가
int main() {
        Triangle tri;
        tri.setWidth(3);
        tri.setHeight(5);
        cout << "삼각형의 면적은 " << tri.getArea() << endl;
        return 0;
}
```

*폭1, 높이1인 삼각형 생성*
*삼각형의 면적은 7.5*
*폭3, 높이5인 삼각형 소멸*

# 실습문제2



- 149~156페이지 문제 중에서 1, 4, 9, 12번을 풀어서 제출하시오.
- 12번은 헤더파일과 소스파일로 분리할 필요 없음, 지금처럼 하나의 소스파일에 작성하면 됨

# 과제제출방법



- 소스코드, 라인단위의 주석, 실행결과를 포함하는 pdf파일을 작성한 후 eclass 과제 게시판에 업로드, **반드시 하나의 pdf파일로 업로드할 것**
- 기한 : 과제 게시판에 마감시간 참조
- 실행결과를 캡쳐할 때 글자를 알아보기 쉽게 확대해서 캡쳐할 것.
- 소스코드의 첫 부분은 아래처럼 제목,날짜,작성자(학번,이름)를 작성할 것

```
// ************************************************
//    제   목 :  정수 4개의 평균을 구하는 프로그램
//    날   짜 :  2023년 9월10일
//    작성자 :  15010101 홍길동
// ************************************************

// 소스코드 작성
```