14강. 템플릿

C++ 프로그래밍

jhhwang@kumoh.ac.kr

컴퓨터공학과 황준하

목차

- 템플릿이란?
- 함수 템플릿
- 클래스 템플릿
- 템플릿의 동작 원리 및 주의 사항

템플릿이란?

- 두 변수의 값을 교환하는 swap 함수를 만든다면?

int 변수 2개를 교환하는 함수

```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

double 변수 2개를 교환하는 함수

```
void swap(double &x, double &y)
{
    double temp = x;
    x = y;
    y = temp;
}
```

CPoint 객체 2개를 교환하는 함수

```
void swap(CPoint &x, CPoint &y)
{
    CPoint temp = x;
    x = y;
    y = temp;
}
```

내용은 모두 동일, 단지 타입만 다름

↓

한 번만 작성하는 방법은 없을까?

↓

템플릿(template)

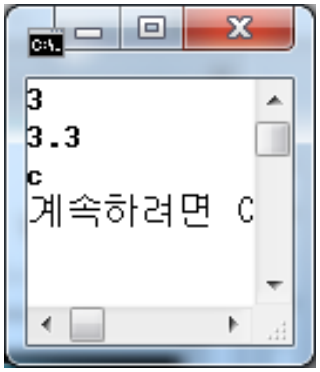
템플릿이란?

- 템플릿
  - 프로그램을 만들어 내는 틀
  - 어떤 타입(int, double, CPoint, ...)에 대해서도 적용이 가능한 함수 또는 클래스의 틀
- 템플릿의 종류
  - 함수 템플릿, 클래스 템플릿
- 표준 템플릿 라이브러리
  - C++ 표준 라이브러리의 일부로서 템플릿을 사용하여 만들어 놓은 라이브러리
  - vector, list, stack, queue 등 → 17강의 주제

함수 템플릿

- Sum 함수 (int, double, char)

```
void main(void)
{
	cout << Sum(1, 2) << endl;
	cout << Sum(1.1, 2.2) << endl;
	cout << Sum('1', '2') << endl;
}
```




```
int Sum(int a, int b)
{
	int c = a + b;
	return c;
}

double Sum(double a, double b)
{
	double c = a + b;
	return c;
}

char Sum(char a, char b)
{
	char c = a + b;
	return c;
}
```

함수 템플릿

- Sum 함수 템플릿

T : int, double 등 특정 타입으로 대치될 문자

```
template <typename T>
T Sum(T a, T b)
{
    T c = a + b;
    return c;
}

void main(void)
{
    cout << Sum(1, 2) << endl;
    cout << Sum(1.1, 2.2) << endl;
    cout << Sum('1', '2') << endl;
}
```

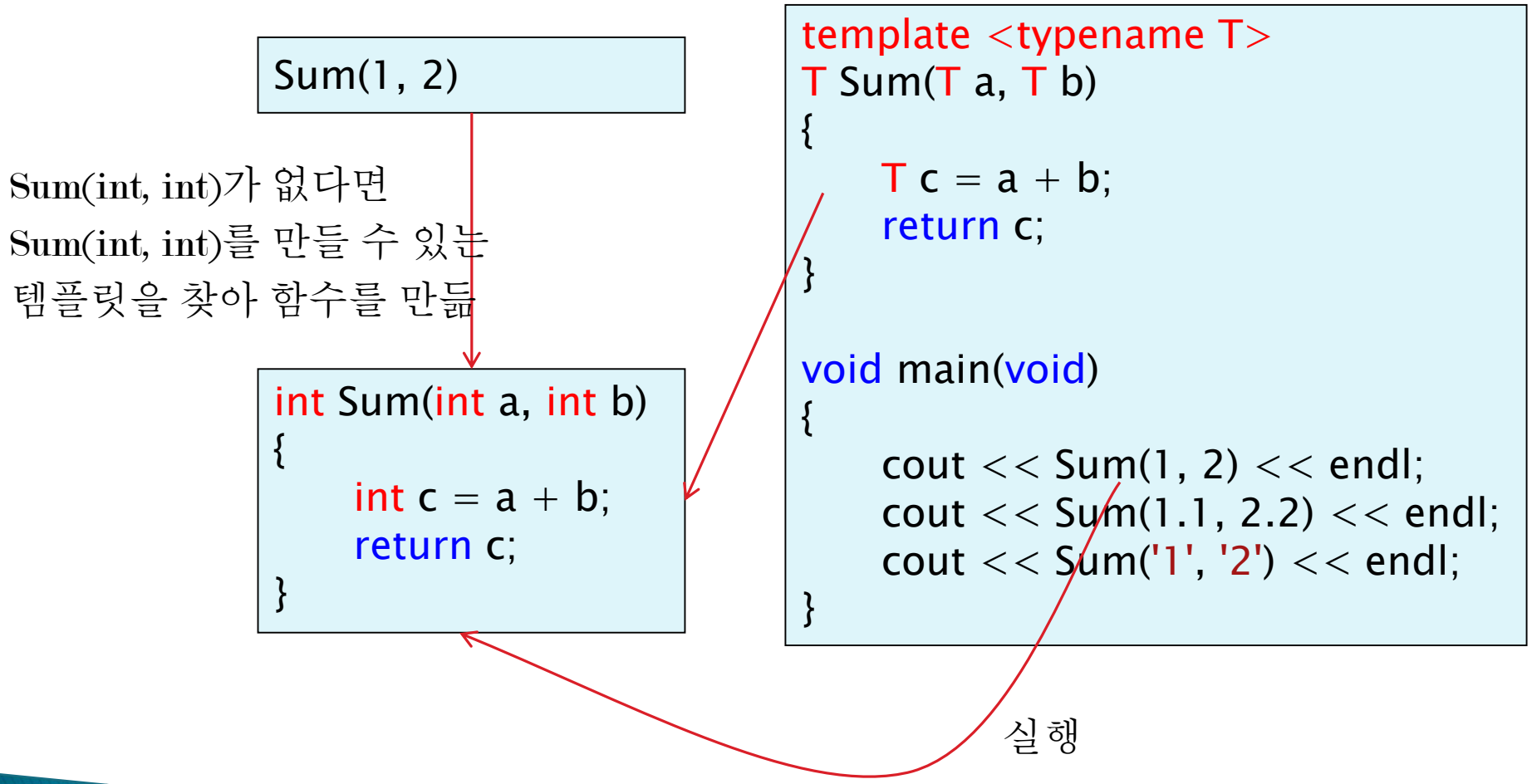
특정 타입으로 대치될 부분만 동일한 문자로 만듦

함수 호출은 이전과 동일

```
int Sum(int a, int b)
{
    int c = a + b;
    return c;
}

double Sum(double a, double b)
{
    double c = a + b;
    return c;
}

char Sum(char a, char b)
{
    char c = a + b;
    return c;
}
```

함수 템플릿

- 동작 원리 : 필요한 실제 함수를 만듦



함수 템플릿 : 연습 문제 (1)

- 어떤 타입에 대해서도 적용 가능한 swap 함수 템플릿을 만들어 보라.

```
void swap(int &x, int &y)
{
      int temp = x;
      x = y;
      y = temp;
}
```

```
void main(void)
{
      int a = 3, b = 4;
      CPoint P1(1, 1), P2(2, 2);

      swap(a, b);
      swap(P1, P2);
}
```

- 프로그램 확인

```
template <typename T>
void swap(T &x, T &y)
{
      T temp = x;
      x = y;
      y = temp;
}
```

함수 템플릿 : 연습 문제 (2)

- 다음 main 함수가 실행될 수 있도록 함수 템플릿 하나를 만들어 보라.

```
void main(void)
{
	CPoint P1;

	Print(1, 2, 3);
	Print("abc", 4, 1.1);
	Print('a', "abc", 2.2);
	Print(5, 6, P1);
}
```

CPoint 클래스의 경우 << 연산자 오버로딩이 되어 있어야 수행 가능

```
template <typename T1, typename T2, typename T3>
void Print(T1 x, T2 y, T3 z)
{
	cout << x << endl;
	cout << y << endl;
	cout << z << endl;
}
```

클래스 템플릿

- 클래스 템플릿
  - 특정 타입의 클래스를 만들어낼 수 있는 클래스의 틀
  - 만드는 방법 → 함수 템플릿과 거의 동일
  - 사용 방법 → 단순 함수 호출과 달리 객체 생성 방법 숙지 필요
- i

```
class CPoint {
private :
	int x, y;

public :
	CPoint(int a, int b) : x(a), y(b) { }
	void Move(int a, int b) { x += a; y += b; }
	void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};
```

클래스 템플릿

- double 좌표를 다루는 클래스

```
class CDoublePoint {
private :
        double x, y;

public :
        CDoublePoint(double a, double b) : x(a), y(b) { }
        void Move(double a, double b) { x += a; y += b; }
        void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};
```

1. 좌표를 나타내는 곳의 타입 변경
2. 클래스명 변경 ← 동일한 클래스 존재 X

클래스명과 타입 외에는 변경 사항 없음

↓

클래스 템플릿으로 만들면 모든 타입에 적용 가능!

클래스 템플릿

- CPoint 클래스 템플릿 만들기

```
template <typename T>
class CPoint {
private :
      T x, y;

public :
      CPoint(T a, T b) : x(a), y(b) { }
      void Move(T a, T b) { x += a; y += b; }
      void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};
```

1. template ... 추가
2. 특정 타입으로 대치될 곳을 T로 변경

주의 사항 : CPoint는 클래스 템플릿의 이름이며 클래스의 이름이 아님

클래스명 : CPoint‹int›, CPoint‹double›, CPoint‹char›, ...

클래스 템플릿

- 클래스 템플릿 사용 방법
  - 클래스명 : CPoint‹int›, CPoint‹double›, ...

```
void main(void)
{
	CPoint<int> P1(1, 2);
	CPoint<double> P2(1.1, 2.2);

	P1.Print();
	P2.Print();
}
```

컴파일러가 int로 대치된 CPoint‹int› 클래스를 만듦

클래스 템플릿

- 클래스 템플릿 멤버 함수의 외부 정의
  - CPoint 클래스 템플릿의 멤버 함수를 외부 정의로 구현한

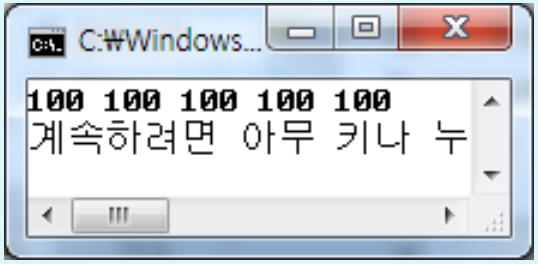
```
template <typename T>
class CPoint {
private :
	T x, y;

public :
	CPoint(T a, T b);
	void Move(T a, T b);
	void Print();
};
```

```
template <typename T>
CPoint<T>::CPoint(T a, T b) : x(a), y(b)
{    }

template <typename T>
void CPoint<T>::Move(T a, T b)
{
	x += a;
	y += b;
}

template <typename T>
void CPoint<T>::Print()
{
	cout << "(" << x << ", " << y << ")" << endl;
}
```

클래스 템플릿 : 연습 문제

- 다음 CArray 클래스는 int 변수 5개를 원소로 갖는 배열을 나타낸 것이다. double 배열, char 배열, 나아가 CPoint 배열도 다룰 수 있도록 클래스 템플릿을 만들어 보라. 함수 내용은 그대로 유지하라.

```
class CArray {
private :
        int ary[5];

public :
        CArray(int a) { for (int i = 0; i < 5; i++) ary[i] = a; }
        void Print() {
                for (int i = 0; i < 5; i++) cout << ary[i] << " ";
                cout << endl;
        }
};
```




```
void main(void)
{
        CArray Ary(100);
        Ary.Print();
}
```

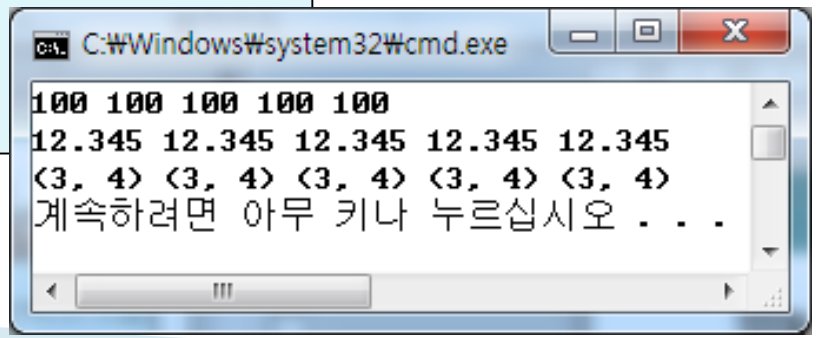
클래스 템플릿 : 연습 문제

- 프로그램 확인

```
template <typename T>
class CArray {
private :
	T ary[5];

public :
	CArray(T a) { for (int i = 0; i < 5; i
	void Print() {
		for (int i = 0; i < 5; i++) cout << ary[i] << " ";
		cout << endl;
	}
};
```

```
void main(void)
{
	CArray<int> Ary1(100);
	Ary1.Print();

	CArray<double> Ary2(12.345);
	Ary2.Print();

	CArray<CPoint> Ary3(CPoint(3, 4));
	Ary3.Print();
}
```

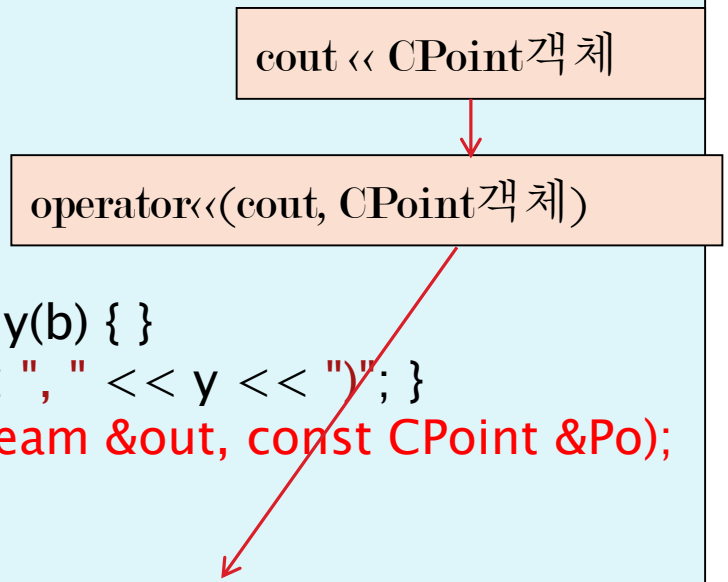
이것이 끝이 아니다. CPoint 객체에 대해 << 연산이 가능하도록 해야 함



클래스 템플릿 : 연습 문제

- 프로그램 확인
  - CPoint 클래스에 대해 << 연산자 오버로딩 추가

```
class CPoint {
private :
	int x, y;

public :
	CPoint(int a = 0, int b = 0) : x(a), y(b) { }
	void Print() { cout << "(" << x << ", " << y << ")"; }
	friend ostream &operator<<(ostream &out, const CPoint &Po);
};

ostream &operator<<(ostream &out, const CPoint &Po)
{
	out << "(" << Po.x << ", " << Po.y << ")";
	return out;
}
```



템플릿의 동작 원리 및 주의 사항

- 함수 (또는 클래스)를 사용하기 위해서는
  - 컴파일 시 : 함수의 프로토타입을 알아야 됨
  - 링크 시 : 프로그램 어딘가에 함수의 정의가 단 한번 나와야 됨
- 템플릿
  - 함수나 클래스를 만들어낼 수 있는 틀
  - 컴파일 시 : 실제 함수 또는 클래스를 만들어 냄
    - 컴파일 : 파일 단위로 수행됨
  - 컴파일 시 템플릿으로부터 실제 함수 몸체까지 만들어야 하므로 템플릿의 모든 내용을 해당 파일 내에 포함하고 있어야만 함

템플릿의 동작 원리 및 주의 사항

- 일반적인 템플릿 사용 방법
  - 헤더 파일에 템플릿의 모든 내용을 포함하고 해당 템플릿을 사용하는 곳에서 이 헤더 파일을 include하여 사용