



9장 템플릿과 가상 함수

변영철 교수

(ycb@jejunu.ac.kr)

제1절 함수 템플릿

- 큰 값(정수, 실수)을 반환하는 함수, `Max`

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int a = Max (2, 3);
```

```
    double b = Max (2.1, 3.4);
```

```
    printf("%d, %f\n", a, b);
```

```
}
```

제1절 함수 템플릿

```
int Max (int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

```
double Max (double a, double b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

제1절 함수 템플릿

- 두 함수 코드가 거의 비슷
- 웬지 여러 번 중복해서 작성하는 느낌
- 템플릿 = 어떤 것(함수)을 만들어 내는(찍어내는) 틀
- 자료 형만 지정하면 함수를 자동으로 만들어 내는 틀 → 함수 템플릿

제1절 함수 템플릿

```
template <typename T>
T Max(T a, T b)
{
    if(a > b)
        return a;
    else
        return b;
}
```

```
#include <stdio.h>

template <typename T>
T Max(T a, T b)
{
    if(a > b)
        return a;
    else
        return b;
}

void main()
{
    printf("%d\n", Max(2, 3));
    printf("%f\n", Max(8.1, 8.8));

    getchar();
}
```

제2절 클래스 템플릿

- 앞의 함수를 클래스 My 로 묶어 추상화하자.
- 그러면 클래스를 찍어내는 클래스 틀로 바뀜.
- 두 값(정수, 실수 등) 중 큰 값을 반환하는 클래스 My

```
#include <stdio.h>
```

```
template <typename T>
```

```
class My {
```

```
public:
```

```
T Max(T a, T b)
```

```
{
```

```
    if(a > b)
```

```
        return a;
```

```
    else
```

```
        return b;
```

```
}
```

```
};
```

```
void main()
```

```
{
```

```
    My<int> gildong;
```

```
    printf("%d\n", gildong.Max(2, 3));
```

```
    My<double> cheolsu;
```

```
    printf("%f\n", cheolsu.Max(8.1, 8.8));
```

```
    getchar();
```

```
}
```

제2절 클래스 템플릿

- 두 수 더하는 클래스 CAdder
- typename 키워드를 class로 해도 됨

```
#include <stdio.h>
```

```
template<class T1, class T2>  
class CAdder
```

```
{  
public:  
    T2 Add(T1 a, T1 b)  
    {  
        T2 sum = a + b;  
        return sum;  
    }  
};
```

```
void main()  
{  
    CAdder<int, long> a;  
  
    printf("%d\n", a.Add(2, 3));  
}
```

아래 코드를 보면 어떤 생각?

```
CAdder<int, long> a;
```

```
CList<int, int> a;
```


제2절 클래스 템플릿

- MFC 라이브러리에 있는 클래스 템플릿 사용하기
 - 프로젝트 이름에서 오른쪽 마우스 버튼을 클릭, 속성 메뉴 항목을 선택, ' 고급 | MFC 사용'에서 '공유 DLL에서 MFC 사용' 선택

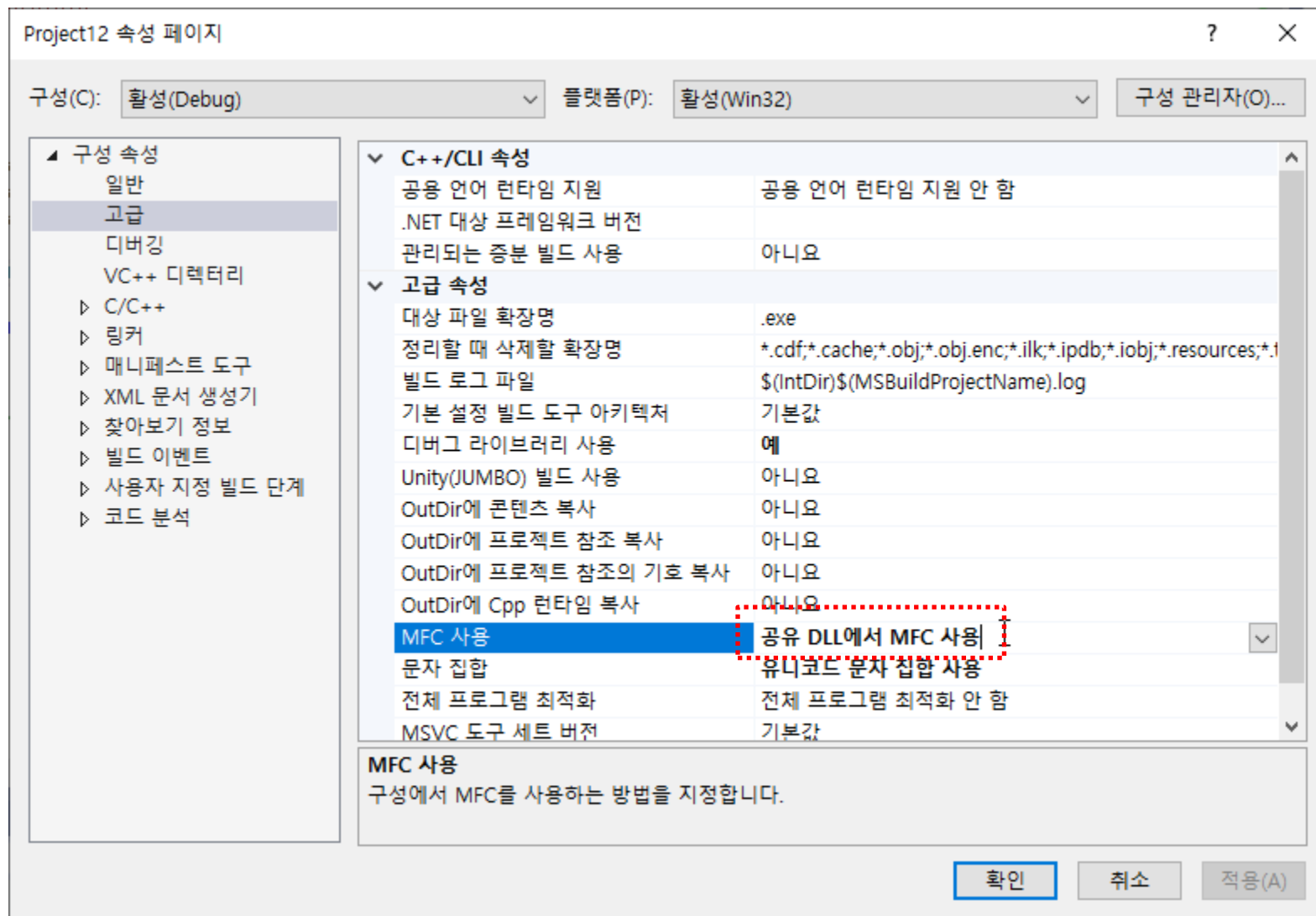
```
#include <afxtempl.h>
#include <stdio.h>
```

```
void main()
{
    CList<int, int> a;

    a.AddTail(20);
    a.AddTail(30);
    a.AddTail(40);

    POSITION pos = a.GetHeadPosition();
    for (int i=0; i < a.GetCount(); i++) {
        printf("%d \n", a.GetNext(pos));
    }
}
```

제2절 클래스 템플릿



제2절 클래스 템플릿

```
#include <afxtempl.h>
#include <stdio.h>
```

```
void ShowList(CList<int, int> & babo) {
    POSITION pos = babo.GetHeadPosition();
    for (int i=0; i < babo.GetCount(); i++) {
        printf("%d \n", babo.GetNext(pos));
    }
}
```




```
void main()
{
    CList<int, int> a;

    a.AddTail(20);
    a.AddTail(30);
    a.AddTail(40);

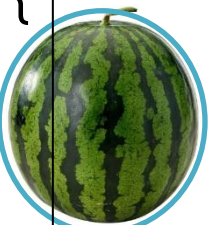


    ShowList(a);
}
```

- ListUtil 클래스로 모듈화하기
- MyList 클래스(Is-a)로 모듈화 하기

제3절 상속과 접근 지정자

class Derived : **public** Base {
 public:
 
 protected:
 
 private:
 
}

public 큰 구멍
protected 중간 구멍
private 작은 구멍

class Base {
 public:
 int a; 
 protected:
 int b; 
 private:
 int c; 
}

제4절 동적할당과 가상 소멸자

- 소멸자 → 객체가 소멸될 때 자동으로 실행되는 멤버 함수

```
#include <stdio.h>
```

```
class CBase  
{  
public:  
    CBase() {  
        printf("CBase \n");  
    }  
    ~CBase() {  
        printf("~CBase \n");  
    }  
};
```

```
void main()  
{  
    CBase gildong;  
}
```

```
CBase  
~CBase  
Press any key to continue
```

실행하면
어떤 결과?
→ 둘 다 실행! 문제없음!!

제4절 동적할당과 가상 소멸자

```
#include <stdio.h>
```

```
class CBase
{
public:
    CBase() {
        printf("CBase %n");
    }
    ~CBase() {
        printf("~CBase %n");
    }
};
```

```
class CDerived : public CBase
{
public:
    CDerived() {
        printf("CDerived %n");
    }

    ~CDerived() {
        printf("~CDerived %n");
    }
};

void main()
{
    CDerived gildong;
}
```

실행하면
어떤 결과?
→ 둘 다 실행! 문제없음!!

제4절 동적할당과 가상 소멸자

- new 연산자로 객체를 동적 할당하여 삭제할 경우 기반 클래스의 소멸자는 실행이 되지 만 파생클래스의 소멸자는 실행되지 않는 문제 발생
- 가상소멸자 : 파생 클래스의 소멸자 함수도 실행되도록 하는 장치

```
void main()
{
    CBase* p = new CDerived;
    delete p; //동적 할당된 객체를 제거
}
```

```
CBase
CDerived
~CBase
Press any key to continue
```

→문제있음!!

제4절 동적할당과 가상 소멸자

- 다음과 같이 virtual로 선언하면 문제가 해결됨
- 기반 클래스의 소멸자를 가상 소멸자로 만들면 파생 클래스의 소멸자가 실행되도록 코드를 작성함

```
class CBase
{
public:
    CBase() {
        printf("CBase %n");
    }

    virtual ~CBase() {
        printf("~CBase %n");
    }
};
```

문제해결!!

제5절 순수 가상 함수와 추상 클래스

- 도형 클래스 CDrawing 클래스를 작성해보자. 여기에 면적을 구하는 멤버함수 area도 작성해 보자.
- 멤버 함수 area를 코딩할 수 있을까?
- 함수 본체를 구현할 수 없는 함수 → 순수 가상 함수
- 순수 가상 함수를 갖는 클래스: 추상 클래스
- 실행되는 몸체가 없으므로 객체를 정의할 수 없음.

```
class CDrawing
{
public:
    double Area()
    {
        //면적을 구하는 공식
        //작성불가!!!
    }
};
```

추상클래스

```
class CDrawing
{
public:
    virtual double Area() = 0;
};
```

순수가상함수

제5절 순수 가상 함수와 추상 클래스

- 객체를 정의하려면 추상 클래스를 상속 받아 새로운 클래스를 작성해야 함 (가령 CCircle)
- 순수 가상 함수(area)는 재정의(overriding)해야 함. 그렇지 않으면 에러가 발생!
- 순수 가상 함수를 사용하는 이유
 - 상속 받는 클래스에서 반드시 재정의하도록 강제
 - 순수 가상 함수를 기술했으므로써 클래스를 **보다 잘 추상화**

```
class CDrawing
{
public:
    virtual double Area() = 0;
};

class CCircle : public CDrawing
{
protected:
    int iRadius;

public:
    double Area()
    {
        return 3.14 * iRadius * iRadius;
    }

    void Set(int r)
    {
        iRadius = r;
    }
};

void main()
{
    CCircle a;
    a.Set(10);
    printf("%f Wn", a.Area());
}
```