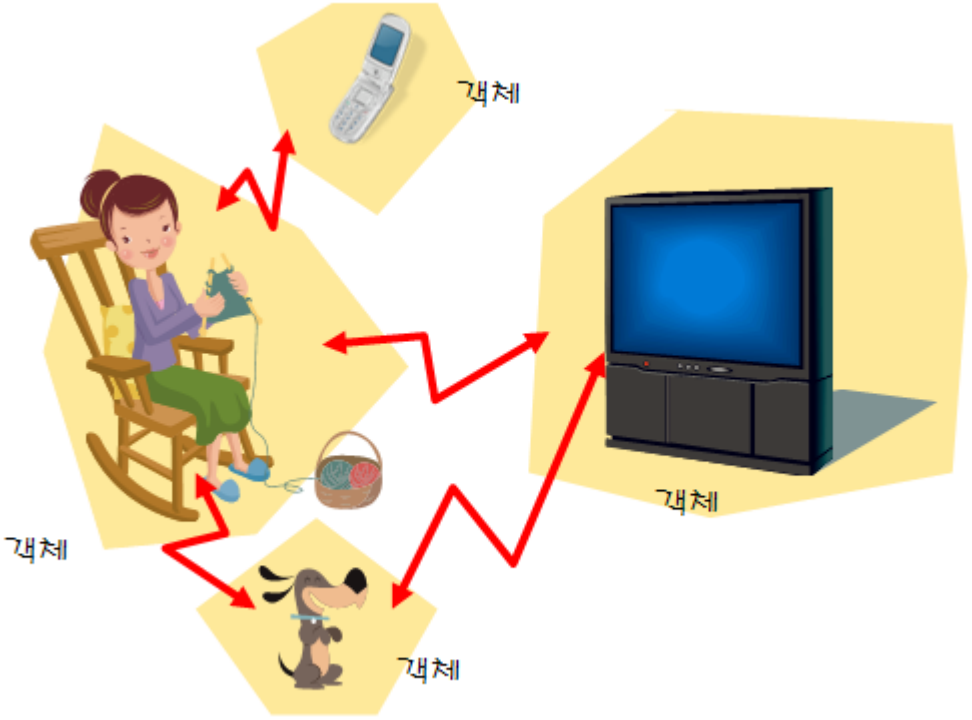
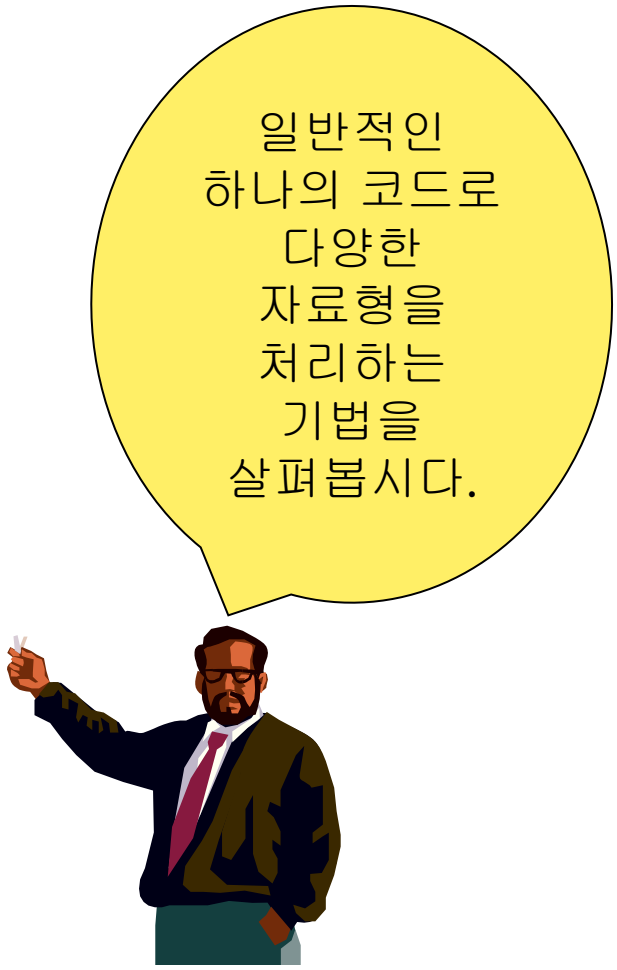
C++ Espresso

# 제12장 템플릿



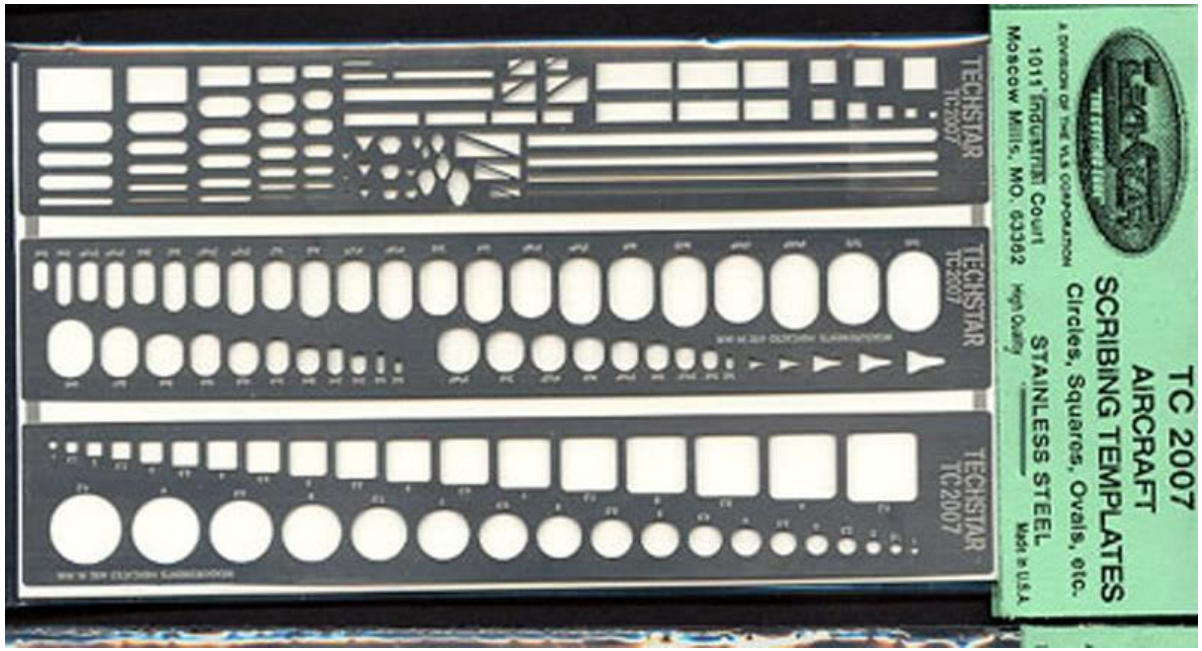
# 이번 장에서 학습할 내용

- 함수 템플릿
- 클래스 템플릿
- 스택 예제



# 템플릿이란?

- 템플릿(template): 물건을 만들 때 사용되는 틀이나 모형을 의미
- 함수 템플릿(function template): 함수를 찍어내기 위한 형틀

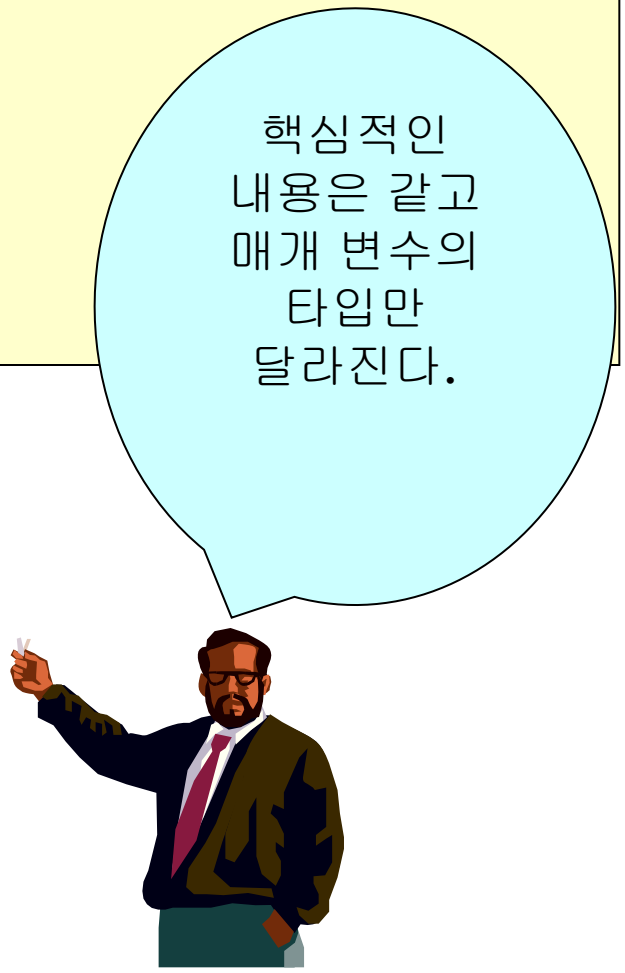


# 함수 get_max()

```
int get_max(int x, int y)
{
    if( x > y ) return x;
    else return y;
}
```

# 함수 get_max()

```
float get_max(float x, float y)
{
    if( x > y ) return x;
    else return y;
}
```



# 일반화 프로그래밍

- 일반화 프로그래밍(generic programming): 일반적인 코드를 작성하고 이 코드를 정수나 문자열과 같은 다양한 타입의 객체에 대하여 재사용하는 프로그래밍 기법


템플릿 함수

```
____ get_max(___x , ___ y)
{
  if( x > y)   return x;
  else return y;
}
```

```
int get_max(int  x , int y)
{
  if( x > y)   return x;
  else return y;
}
```

# get_max()

```
template<typename T>
T get_max(T x, T y)
{
    if( x > y ) return x;
    else return y;
}
```

자료형이 변수처럼 표기되어 있음을 알 수 있다

# 템플릿 함수의 사용

get_max(1, 3) 으로 호출

```
template <typename T>
T get_max(T x, T y)
{
  if(x > y)   return x;
  else return y;
}
```

```
int get_max(int x, int y)
{
  if(x > y)   return x;
  else return y;
}
```

get_max(1.8, 3.7) 으로 호출

```
double get_max(double x, double y)
{
  if(x > y)   return x;
  else return y;
}
```

# 예제

*get_max.cpp*

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

template <typename T>
T get_max(T x, T y)
{
  if(x > y)    return x;
  else return y;
}

int main()
{
  // 아래의 문장은 정수 버전 get_max()를 호출한다.
  cout << get_max(1, 3) << endl;

  // 아래의 문장은 실수 버전 get_max()를 호출한다.
  cout << get_max(1.2, 3.9) << endl;

  return 0;
}
```

# 실행 결과

실행 결과

```
3
3.9
계속하려면 아무 키나 누르십시오 . . .
```

# 템플릿 함수의 특수화

```
template <typename T>                    // 함수 템플릿으로 정의
void print_array(T[] a, int n)
{
    for(int i=0;i<n; i++)
        cout << a[i] << " ";
    cout << endl;
}

template <>                              // 템플릿 특수화
void print_array(char[] a, int n)        // 매개 변수가 char인 경우에는 이 함수가 호출된다.
{
    cout << a << endl;
}
```

# 함수 템플릿과 함수 중복

*swap_values.cpp*

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

template <typename T>
void swap_values(T& x, T& y)
{
   T   temp;
   temp = x;
   x = y;
   y = temp;
}
void swap_values(char* s1, char* s2)
{
   int   len;

   len = (strlen(s1) >= strlen(s2)) ? strlen(s1) : strlen(s2);
   char* tmp = new char[len + 1];

   strcpy(tmp, s1);
   strcpy(s1, s2);
   strcpy(s2, tmp);
   delete[] tmp;
}
```

템플릿함수

함수 중복

# 함수 템플릿과 함수 중복

```
int main()
{
   int x=100, y=200;
   swap_values(x, y);              // x, y가 모두 int 타입- OK!
   cout << x << " " << y << endl;

   char s1[100]="This is a first string";
   char s2[100]="This is a second string";
   swap_values(s1, s2);            // s1, s2가 모두 배열 - 오버로딩 함수 호출
   cout << s1<< " " << s2<< endl;
   return 0;
}
```

---

실행 결과

```
200 100
This is a second string This is a first string
계속하려면 아무 키나 누르십시오 . . .
```

# 두개의 타입 매개 변수

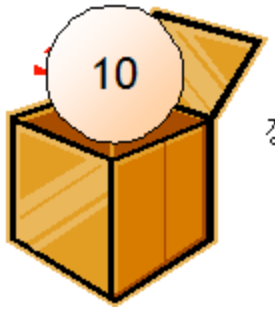
```
template<typename T1, typename T2>
void copy(T1 a1[], T2 a2[], int n)
{
    for (int i = 0; i < n; ++i)
        a1[i] = a2[i];
}
```

# 클래스 템플릿

- 클래스 템플릿(class template): 클래스를 찍어내는 틀(template)

```
template <typename 타입이름, ...> class 클래스이름
{
...

}
```

- 예제: 하나의 값을 저장하고 있는 박스




정수를 저장하는 상자

# 예 제

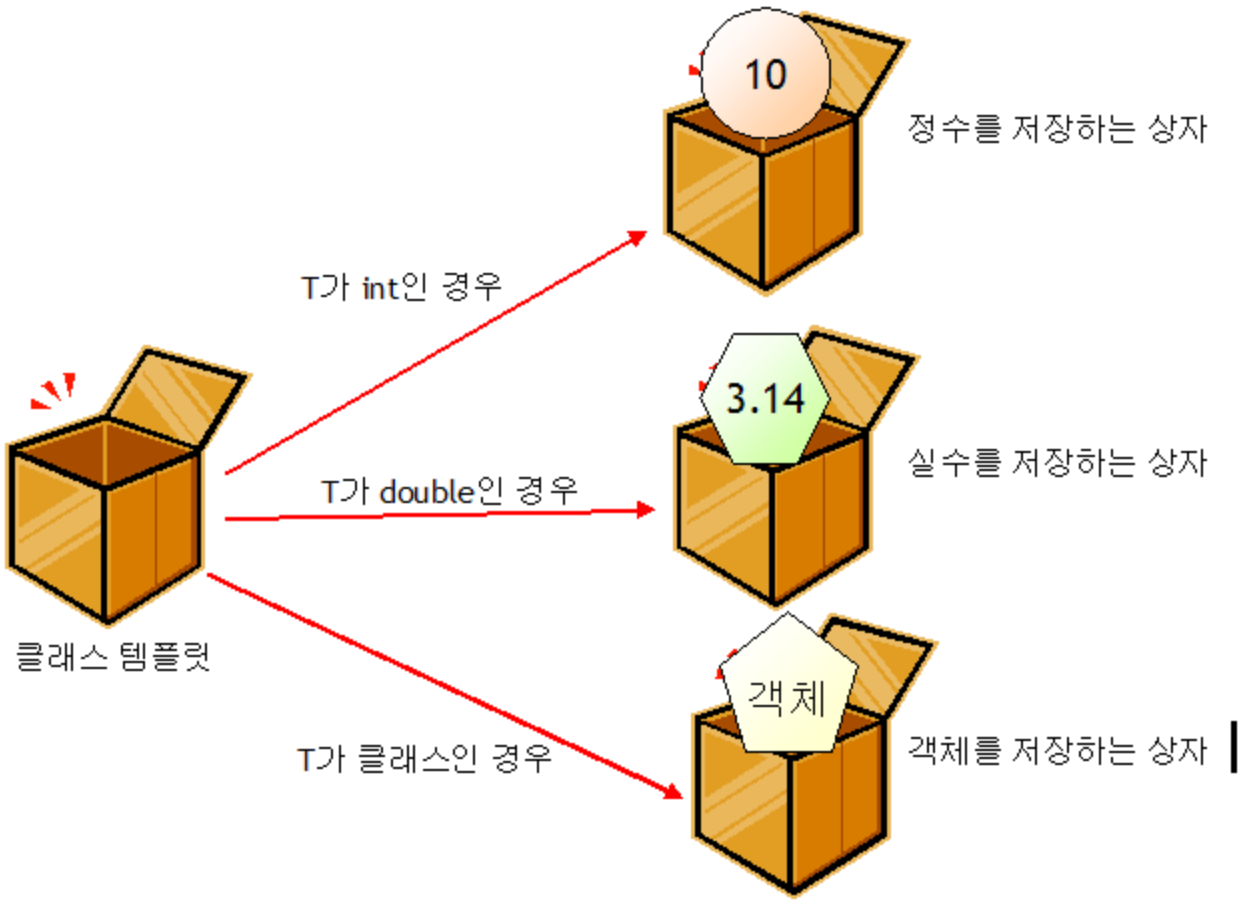
```
class Box {
    int data;
public:
    Box() { }
    void set(int value) {
        data = value;
    }
    int get() {
        return data;
    }
};

int main()
{
   Box box;
   box.set(100);
   cout << box.get() << endl;
   return 0;
}
```

# 실행 결과

실행 결과

```
100
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 템플릿 버전



# 클래스 템플릿 정의

```
template <typename T>
class 클래스이름
{
...// T를 어디서든지 사용할 수 있다.
}
```

# 예 제

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

template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸다.
public:
    Box() {      }
    void set(T value) {
        data = value;
    }
    T get() {
        return data;
    }
};
```

클래스 템플릿

```
int main()
{
   Box<int> box;
   Box.set(100);
   cout << box.get() <<endl;

   Box<double> box1;
   box1.set(3.141592);
   cout << box1.get() <<endl;

   return 0;
}
```

정수버전 클래스

실수 버전 클래스

실행 결과

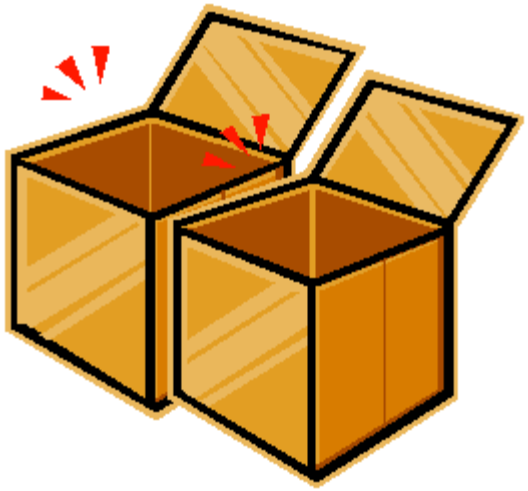
```
100
3.14159
계속하려면 아무 키나 누르십시오 . . .
```

# 클래스 외부에서 구현

```
template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸다.
public:
    Box();
    void set(T value);
    T get();
};

template <typename T>
Box<T>::Box() {
}

template <typename T>
void Box<T>::set(T value) {
  data = value;
}

template <typename T>
T Box<T>::get() {
  return data;
}
```

# 두개의 타입 매개 변수

- 두 개의 데이터를 저장하는 클래스 Box2



Box2 클래스 템플릿

# 예 제

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

template <typename T1, typename T2>
class Box2 {

   T1 first_data; // T1은 타입(type)을 나타낸다.
   T2 second_data; // T2는 타입(type)을 나타낸다.

public:
   Box2() {      }
   T1 get_first();
   T2 get_second();
   void set_first(T1 value) {
        first_data = value;
   }
   void set_second(T2 value) {
        second_data = value;
   }
};
```

두개의 타입 매개변수를 가지는 클래스 템플릿

# 예제

```
template <typename T1, typename T2>
T1 Box2<T1, T2>::get_first() {
   return first_data;
}
template <typename T1, typename T2>
T2 Box2<T1, T2>::get_second() {
   return second_data;
}
int main()
{
   Box2<int, double> b;
   b.set_first(10);
   b.set_second(3.14);
   cout << "(" << b.get_first() << ", " << b.get_second() << ")" << endl;
   return 0;
}
```

---

실행 결과

```
(10, 3.14)
계속하려면 아무 키나 누르십시오 . . .
```