14장 예외처리와 템플릿(2)

2020. 12. 3

순천향대학교 컴퓨터 공학과

내용

- 제네릭 프로그래밍이란?
- 템플릿 함수
- 중복함수와 템플릿 함수
- 템플릿 클래스
- C++ STL

중복 함수는 코드 중복 초래

```
void myswap(int& a, int& b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
}
void myswap(double & a, double & b) {
  double tmp;
  tmp = a;
  a = b;
  b = tmp;
}
```

코드 중복

```
int main() {
  int a=4, b=5;
  myswap(a, b);
  cout << a << '\t' << b << endl;

  double c=0.3, d=12.5;
  myswap(c, d);   cout << c << '\t' << d <<
endl;
}
```

중복 함수의 문제점 해결 방안은?

```
void myswap(int & a, int & b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
}
```

```
void myswap (double & a, double & b)
{
  double tmp;
  tmp = a;
  a = b;
  b = tmp;
}
```

제네릭 함수 만들기
(일반화)

```
template <class T>
void myswap (T & a, T &
b) {
  T tmp;
  tmp = a;
  a = b;
  b = tmp;
}
```

제네릭 프로그래밍이란?

- generic programming
  - 'generic'의 사전적 의미는 '비슷한 것들을 공유하는'
  - 포괄적 또는 일반화 프로그래밍이라고도 부름
  - 템플릿 함수나 템플릿 클래스를 활용하는 프로그래밍
  - C++ STL(Standard Template Library)
- 함수 코드의 재사용 장점
  - 높은 소프트웨어의 생산성과 유용성
- 보편화 추세
  - Java, C# 등 많은 언어에서 지원

템플릿 함수

- C++에서 하나의 형틀을 만들어서 다양한 코드를 생산해 내도록 할 수 있는데 이것을 템플릿이라고 한다. 템플릿 함수(function template)는 함수를 찍어내기 위한 형틀이다.

템플릿함수

```
____ 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;
}
```

템플릿 함수 작성

- C++ 템플릿 함수 작성
  - template 키워드로 타입 매개변수 선언
  - 타입 매개변수를 매개변수로 갖는 템플릿 함수 작성

```
template <class T> 또는
template <typename T>
```

타입 매개변수 선언

```
template <class T>
void myswap (T & a, T & b) {
   T tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

템플릿 함수 myswap

템플릿 함수 사례화

- 컴파일러는 템플릿 함수를 호출문의 실 매개변수 타입별로 실제 함수를 생성

```
template <class T>
void myswap (T & a, T & b)
{
   T tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

```
int a, b;
double c, d;
...
myswa[(a, b);

myswap(c, d);
```

사례화

```
void myswap (int & a, int & b) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

```
void myswap (double & a, double & b) {
   double tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

예제

```
#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()
{
        cout << get_max(1, 3) << endl;
        cout << get_max(1.2, 3.9) << endl;
        return 0;
}
```

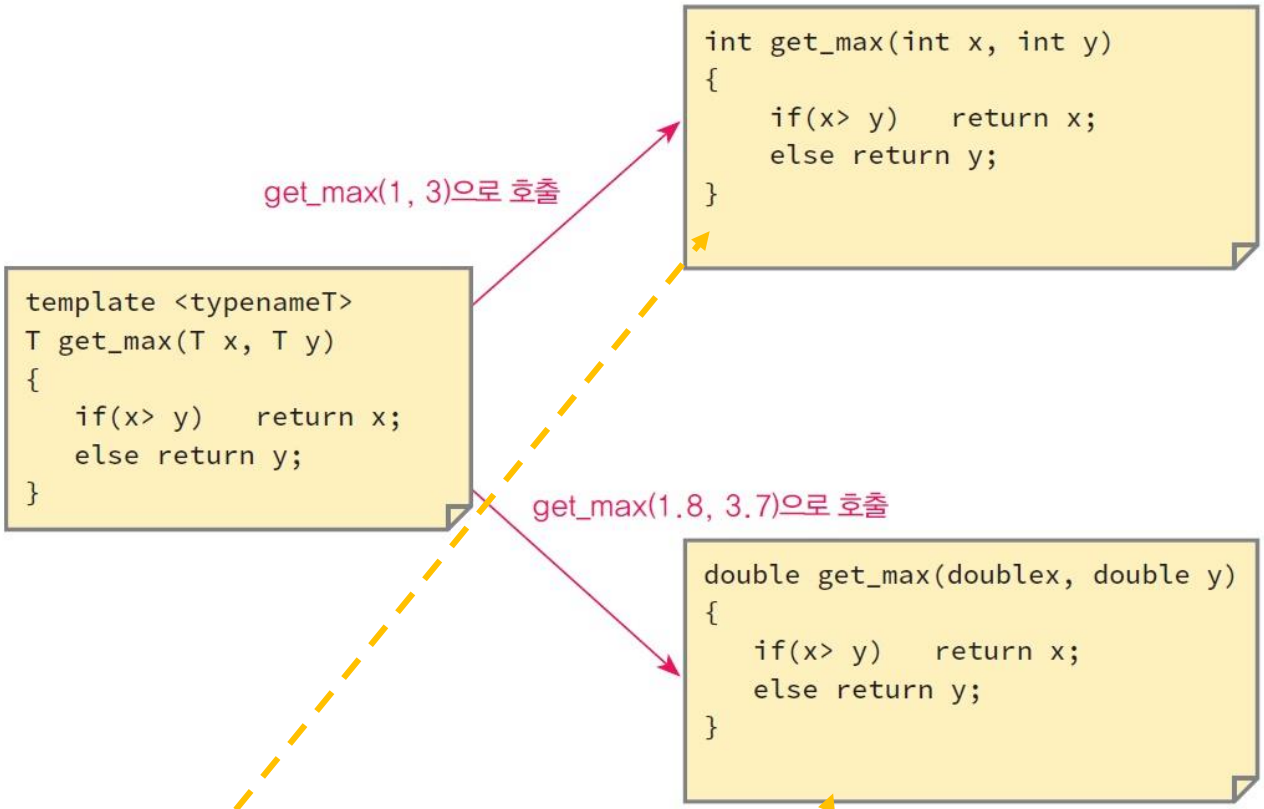


그림 14.7 템플릿 함수

예제: 템플릿 함수

```
template<typename T>
void copy_array(T a[], T b[], int n)
{
        for (int i = 0; i < n; ++i)
                a[i] = b[i];
}
```

```
template <typename T>
T get_first(T[] a)
{
        return a[0];
}
```

예제

- 다음 코드 고려:

```
template <class T>
void myswap (T & a, T & b) {
  T tmp;
  tmp = a;
  a = b;
  b = tmp;
}
```

```
int s=4;
double t=5;

myswap(s, t);  // ?
```

타입 매개변수가 2개인 경우

- 타입 매개변수를 2개 이상 선언 가능

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

예제

```
template <class T1, class T2>
void mcopy(T1 src [], T2 dest [], int n) {
  for(int i=0; i<n; i++)
    dest[i] = (T2)src[i];
}

int main() {
  int x[] = {1,2,3,4,5};
  double d[5];
  char c[5] = {'H', 'e', 'l', 'l', 'o'}, e[5];

  mcopy(x, d, 5);
  mcopy(c, e, 5);

  for(int i=0; i<5; i++) cout << d[i] << ' ';
  cout << endl;

  for(int i=0; i<5; i++) cout << e[i] << ' ';
  cout << endl;
}
```

중복 함수와 템플릿 함수

- 중복된 함수의 본체가 모두 동일한 경우에는 템플릿 함수로 작성하는 것이 바람직

중복된 함수들

```
void myswap(int & a, int & b) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

```
void myswap (double & a, double & b) {
   double tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

템플릿 함수

```
template <class T>
void myswap (T & a, T & b)
{
   T tmp;
   tmp = a;
   a = b;
   b = tmp;
}
```

템플릿 함수보다 중복 함수가 우선

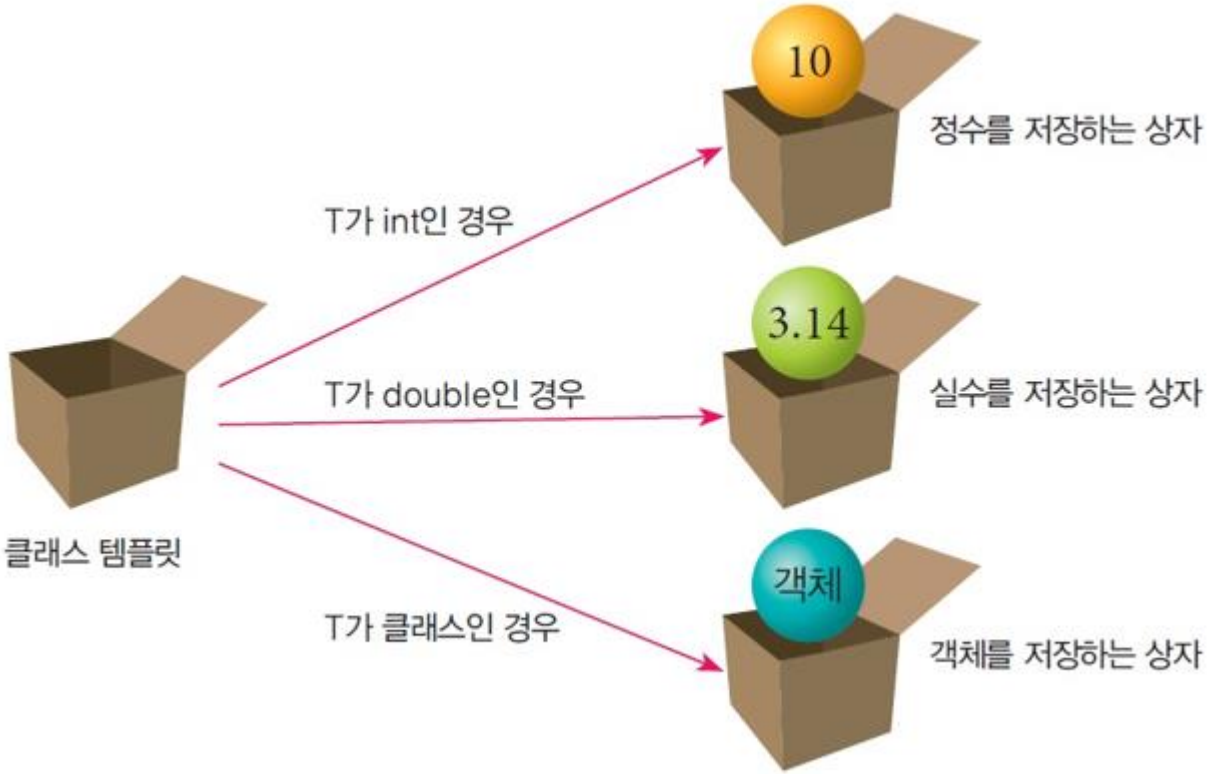
- 다음 코드의 실행 결과는?

```
template <class T>
void print(T array [], int n) {
   for(int i=0; i<n; i++)
      cout << array[i] << '₩t';
   cout << endl;
}

void print(char array [], int n) {
   for(int i=0; i<n; i++)
      cout << (int)array[i] << '₩t';
   cout << endl;
}

int main() {
   int x[] = {1,2,3,4,5};
   double d[5] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    char c[5] = {1,2,3,4,5};

   print(x, 5);
   print(d, 5);
   print(c, 5);
}
```

템플릿 함수와 이름이 동일한 함수가 중복될 경우에, 중복된 함수가 템플릿 함수보다 우선하여 바인딩함

템플릿 클래스

- template를 이용하여 템플릿 클래스 작성 가능



템플릿 클래스 작성

- 템플릿 클래스 선언부와 구현부 모두 template로 선언해야 함
  - 템플릿 클래스의 멤버 함수는 자동으로 템플릿 함수

```
template <class T>
class MyStack {
   int tos;
   T data [100];
public:
   MyStack();
   void push(T element);
   T pop();
};
```

```
template <class T>
void MyStack<T>::push(T element) {
   ...
}
template <class T>
T MyStack<T>::pop() {
   ...
}
```

예제

```
template <typename T>
class Box {
        T data;
public:
        Box() { }
        void set(T value) {
                data = value;
        }
        T get() {
                return data;
        }
};
```

```
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;
}
```

예제

```
template <class T>
class MyStack {
   int tos;// top of stack
   T data [100];
public:
   MyStack();
   void push(T element);
   T pop();
};

template <class T>
MyStack<T>::MyStack() {
   tos = -1;
}

template <class T>
void MyStack<T>::push(T element) {
   if(tos == 99) {
      cout << "stack full";
      return;
   }
   tos++;
   data[tos] = element;
}
```

```
template <class T>
T MyStack<T>::pop() {
   T retData;
   if(tos == -1) {
      cout << "stack empty";
      return 0; // 오류
   }
   retData = data[tos--];
   return retData;
}

int main() {
   MyStack<int> iStack; // int 타입 스택 객체 생성
   iStack.push(3);
   cout << iStack.pop() << endl;

   MyStack<double> dStack; // double 타입 스택 객체 생성
   dStack.push(3.5);
   cout << dStack.pop() << endl;

   MyStack<char> *p = new MyStack<char>();
   p->push('a');
   cout << p->pop() << endl;
   delete p;
}
```

타입 매개 변수가 2개 이상인 경우

- 템플릿 클래스는 2개 이상의 타입 매개변수 포함 가능
  - 형식: template<class T1, class T2>

예제

```
template <typename T1, typename T2>
class Box2 {
        T1 first_data;
        T2 second_data;
public:
        Box2() { }
        T1 get_first();
        T2 get_second();
        void set_first(T1 value) {
                first_data = value;
        }
        void set_second(T2 value) {
                second_data = value;
        }
};
```

```
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;
}
```

예제

```
template <class T1, class T2>
class GClass {
   T1 data1;
   T2 data2;
public:
   GClass();
   void set(T1 a, T2 b);
   void get(T1 &a, T2 &b);
};

template <class T1, class T2>
GClass<T1, T2>::GClass() {
   data1 = 0; data2 = 0;
}

template <class T1, class T2>
void GClass<T1, T2>::set(T1 a, T2 b) {
   data1 = a; data2 = b;
}

template <class T1, class T2>
void GClass<T1, T2>::get(T1 & a, T2 & b) {
   a = data1; b = data2;
}
```

```
int main() {
   int a;
   double b;
   GClass<int, double> x;
   x.set(2, 0.5);
   x.get(a, b);
   cout << "a=" << a << '₩t' << "b=" << b << endl;

   char c;
   float d;
   GClass<char, float> y;
   y.set('m', 12.5);
   y.get(c, d);
   cout << "c=" << c << '₩t' << "d=" << d << endl;
}
```

C++ 표준 템플릿 라이브러리, STL

- STL(Standard Template Library)
  - 다양한 템플릿 클래스와 템플릿 함수 포함
- STL의 구성
  - 컨테이너 – 템플릿 클래스
    - 데이터를 담아두는 자료 구조를 표현한 클래스
    - 리스트, 큐, 스택, 맵, 셋, 벡터
  - iterator – 컨테이너 원소에 대한 포인터
    - 컨테이너의 원소들을 접근하기 위한 포인터
  - 알고리즘 – 템플릿 함수
    - 복사, 검색, 삭제, 정렬 등의 기능을 구현한 템플릿 함수

C++ STL

- STL 컨테이너 종류

컨테이너 클래스	설명	헤더 파일
vector	가변 크기의 배열을 일반화한 클래스	<vector>
deque	앞뒤 모두 입력 가능한 큐 클래스	<deque>
list	빠른 삽입/삭제 가능한 리스트 클래스	<list>
set	정렬된 순서로 값을 저장하는 집합 클래스. 값은 유일	<set>
map	(key, value) 쌍을 저장하는 맵 클래스	<map>
stack	스택을 일반화한 클래스	<stack>
queue	큐를 일반화한 클래스	<queue>

C++ STL (2)

- STL 알고리듬

copy	merge	random	rotate
equal	min	remove	search
find	move	replace	sort
max	partition	reverse	swap