12장. 템플릿

부산대학교
ddosun@pusan.ac.kr
김미경

12-1 템플릿(template)에 대한 이해

- **IntroTemplate1.cpp**

```
int Add(int a, int b)
{
    return a+b;
}
```

템플릿화

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

```
1  /*    IntroTemplate1.cpp    */
2  #include <iostream>
3  using std::endl;using std::cout;
4
5  template <typename T>
6  T Add(T a, T b)
7  {
8      return a+b;
9  }
10
11 int main(void)
12 {
13     cout<<Add(10, 20)<<endl;
14     cout<<Add(1.1, 2.2)<<endl;
15
16     return 0;
17 }
```

string 객체는 Add될까?

12-2 함수 템플릿

- 함수 템플릿
- 둘 이상의 타입에 대해서 템플릿화
  - IntroTemplate2_error.cpp➔ IntroTemplate3.cpp

```
/////*   IntroTemplate2.cpp  */
#include <iostream>
using std::endl;
using std::cout;

template <typename T> // 함수 템플릿 정의
void ShowData(T a, T b)
{
    cout<<a<<endl;
    cout<<b<<endl;
}

int main(void)
{
    ShowData(1, 2);
    ShowData(3, 2.5);

    return 0;
}
```

```
///* IntroTemplate3.cpp  */
#include <iostream>
using std::endl;
using std::cout;

 // 함수 템플릿 정의
template <typename T1, typename T2>
void ShowData(T1 a, T2 b)
{
    cout<<a<<endl;
    cout<<b<<endl;
}

int main(void)
{
    ShowData(1, 2);
    ShowData(3, 2.5);

    return 0;
}
```

12-2 함수 템플릿

- **함수 템플릿의 특수화**
  - **SepciFuncTemplate2**

```
template <typename T>
int SizeOf(T a)
{
    return sizeof(a);
}
```

특수화 →

```
template<>
int SizeOf(char* a)
{
    return strlen(a);
}
```

```
/////*   SepciFuncTemplate1.cpp   *////
#include <iostream>
using std::endl;using std::cout;

template <typename T> // 함수 템플릿 정
int SizeOf(T a)
{
    return sizeof(a);
}
int main(void)
{
    int i=10;
    double e=7.7;
    char* str="Good morning!";

    cout<<SizeOf(i)<<endl;
    cout<<SizeOf(e)<<endl;
    cout<<SizeOf(str)<<endl;

    return 0;
}
```

```
//////*   SepciFuncTemplate2.cpp   *////
#include <iostream>
#include <cstring>
using std::endl;using std::cout;

template <typename T> // 함수 템플릿 정의
int SizeOf(T a)
{
    return sizeof(a);
}
template<> // 함수 템플릿 특수화 정의
int SizeOf(char* a)//전달인자가 char*인 경우 이 함수 호출
{
    return strlen(a);
}
int main(void)
{
    int i=10;
    double e=7.7;
    char* str="Good morning!";

    cout<<SizeOf(i)<<endl;
    cout<<SizeOf(e)<<endl;
    cout<<SizeOf(str)<<endl;
    return 0;
}
```

12-3 클래스 템플릿

```
class Data
{
    int data;
public:
    Data(int d){  data=d; }
    void SetData(int d){
        data=d;
    }
    int GetData(){
        return data;
    }
};
```

템플릿화

```
template <typename T>
class Data
{
    T data;
public:
    Data(T d){ data=d; }
    void SetData(T d){
        data=d;
    }
    T GetData(){
        return data;
    }
};
```

```
int main(void)
{
   Data<int> d1(0);
   d1.SetData(10);
   Data<char> d2('a');
   . . . . .
```

12-3 클래스 템플릿

```
template <typename T>
class Data
{
    T data;
public:
    Data(T d){ data=d; }
    void SetData(T d){
        data=d;
    }
    T GetData(){
        return data;
    }
};
```

선언/정의 분리

```
template <typename T>
class Data
{
    T data;
public:
    Data(T d);
    void SetData(T d);
    T GetData();
};

template <typename T>
Data<T>::Data(T d){
    data=d;
}
template <typename T>
void Data<T>::SetData(T d){
    data=d;
}
template <typename T>
T Data<T>::GetData(){
    return data;
}
```

클래스 템플릿

클래스 템플릿으로 만들어진 class
➔템플릿 클래스
템플릿 클래스 생성되는 코드를 컴파일 할 때
➔.h와 .cpp 분리시키면 안 됨➔.h에 둠

12-3 클래스 템플릿

```
1  /*   DataTemplate2.cpp*/
2  #include <iostream>
3  using std::endl;using std::cout;
4
5  template <typename T>
6  class Data{
7      T data;
8  public:
9      Data(T d){
10         data=d;
11     }
12     void SetData(T d){
13         data=d;
14     }
15     T GetData(){
16         return data;
17     }
18 };
19 int main(void){
20     Data<int> d1(0);
21     d1.SetData(10);
22
23     Data<char> d2('a');
24
25     cout<<d1.GetData()<<endl;
26     cout<<d2.GetData()<<endl;
27 }
```

```
1  /*   DataTemplate3.cpp*/
2  #include <iostream>
3  using std::endl;using std::cout;
4
5  template <typename T>
6  class Data
7  {
8      T data;
9  public:
10     Data(T d);
11     void SetData(T d);
12     T GetData();
13 };
14 template <typename T>
15 Data<T>::Data(T d){
16     data=d;
17 }
18 template <typename T>
19 void Data<T>::SetData(T d){
20     data=d;
21 }
22 template <typename T>
23 T Data<T>::GetData(){
24     return data;
25 }
```

Main동일

12-4 템플릿의 원리 이해

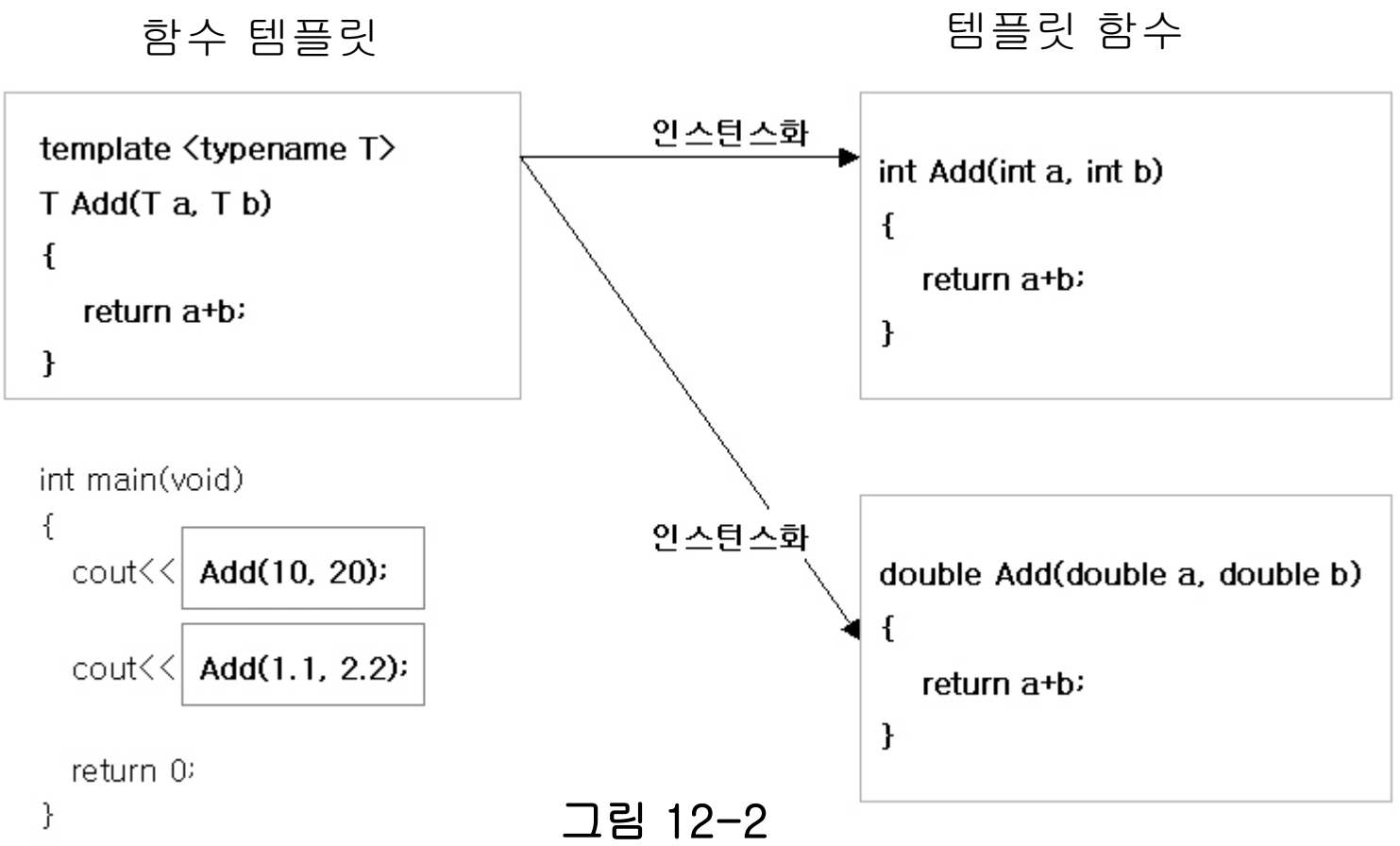

그림 12-2

템플릿 함수의 코드가 생성되는 시기는 코드를 컴파일 할 때
➔컴파일 시기 오래 걸림, but 유연한 코드

12-5. 스택 템플릿

```
1  /*   StackTemplate1.cpp*/
2  #include <iostream>
3  using std::cout;using std::endl;
4  class Stack {
5  private:
6     int topIdx;   // 마지막 입력된 위치의 인덱스.
7     char* stackPtr;  // 스택 포인터.
8  public:
9      Stack(int s=10);
10     ~Stack();
11     void Push(const char& pushValue);
12     char Pop();
13 };
14
15 Stack::Stack(int len){
16     topIdx=-1;      // 스택 인덱스 초기화.
17     stackPtr=new char[len]; // 데이터 저장 위한 배열 선언.
18 }
19 Stack::~Stack(){
20     delete[] stackPtr;
21 }
22 void Stack::Push(const char& pushValue){ // 스택에 데이터 입
23     stackPtr[++topIdx]=pushValue;
24 }
25 char Stack::Pop(){ // 스택에서 데이터 꺼냄.
26     return stackPtr[topIdx--];
27 }
```

```
29 int main()
30 {
31     Stack stack(10);
32     stack.Push('A');
33     stack.Push('B');
34     stack.Push('C');
35
36     for(int i=0; i<3; i++){
37         cout<<stack.Pop()<<endl;
38     }
39
40     return 0;
41 }
```

12-5. 스택 템플릿-여러 자료 저장

```
1  /*   StackTemplate2.cpp*/
2  #include <iostream>
3  using std::cout;using std::endl;
4  template <typename T>
5  class Stack {
6  private:
7      int topIdx;   // 마지막 입력된 위치의 인덱스.
8     T* stackPtr;  // 스택 포인터.
9  public:
10     Stack(int s=10);    ~Stack();
11     void Push(const T& pushValue);  T Pop();
12 };
13 template <typename T>
14 Stack<T>::Stack(int len){
15     topIdx=-1;      // 스택 인덱스 초기화.
16     stackPtr=new T[len];    // 데이터 저장 위한 배열 선언.
17 }
18 template <typename T>
19 Stack<T>::~Stack(){
20     delete[] stackPtr;
21 }
22 template <typename T>
23 void Stack<T>::Push(const T& pushValue){ // 스택에 데이터 입력.
24     stackPtr[++topIdx]=pushValue;
25 }
26 template <typename T>
27 T Stack<T>::Pop(){ // 스택에서 데이터 꺼냄.
28     return stackPtr[topIdx--];
29 }
```

```
32 int main()
33 {
34     Stack<char> stack1(10);
35     stack1.Push('A');
36     stack1.Push('B');
37     stack1.Push('C');
38
39     for(int i=0; i<3; i++){
40         cout<<stack1.Pop()<<endl;
41     }
42
43     Stack<int> stack2(10);
44     stack2.Push(10);
45     stack2.Push(20);
46     stack2.Push(30);
47
48     for(int j=0; j<3; j++){
49         cout<<stack2.Pop()<<endl;
50     }
51
52     return 0;
53 }
```

12.6. 표준 템플릿 라이브러리(STL)

- STL : 템플릿을 사용해서 만들어진 표준화된 라이브러리
- 라이브러리
  - 클래스나 함수를 모아놓은 것
- STL
  - 일반적으로 많이 사용하는 클래스와 함수
  - (예) 링크드 리스트 클래스, 동적 배열 클래스, 정렬 함수, 검색 함수 등과 같이 범용적인 클래스와 함수
    - 템플릿으로 만들어져 있기 때문에 확장이 용이.

12-6. 표준 템플릿 라이브러리(STL)

- STL을 사용함으로써 얻을 수 있는 장점
  - STL은 표준➔누구나 똑같은 클래스와 함수를 사용한
  - ➔ 다른 사람이 작성해 놓은 코드를 쉽게 이해
  - STL은 전문가➔직접 만든 것보다 효율적, 안전

list

```
1  /* list 예제.cpp */
2  #include <list>
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8    // int 타입을 담을 링크드 리스트 생성
9    list<int> intList;
10
11     // 1 ~ 10까지 링크드 리스트에 넣는다.
12     for (int i = 0; i < 10; ++i)
13         intList.push_back(i);
14
15     // 5를 찾아서 제거한다.
16      intList.remove( 5);
17
18     // 링크드 리스트의 내용을 출력한다.
19     list<int>::iterator it;
20
21     for (it = intList.begin(); it != intList.end(); ++it)
22         cout << *it << "\t";
23
24     return 0;
25   }
```