

# 객체지향언어



## 12장 일반화 프로그래밍

# 학습목표

- 일반화된 함수를 만들기 위한 템플릿 함수를 사용할 수 있다.
- 일반화된 클래스를 정의하는 템플릿 클래스를 사용할 수 있다.
- `string` 클래스의 문자열을 사용할 수 있다.
- 표준 템플릿 라이브러리를 설명할 수 있다.

## ■ 일반화 프로그래밍

- C++에서 제공하는 템플릿을 사용하면 자료형에 관계없는 일반형으로 함수뿐만 아니라 클래스까지도 만들 수 있음
- 템플릿 함수와 클래스에 적용할 자료형은 함수나 클래스를 호출할 때 컴파일러가 결정
- C++는 템플릿을 사용하여 만들어진 여러 가지의 유용한 함수와 클래스를 STL(standard template library)로 제공
- STL은 일반화한 자료구조 라이브러리로 이미 검증을 마친 함수와 클래스로, STL을 이용하면 오류 없는 프로그램을 빠른 시일 내에 쉽게 개발할 수 있음

## ■ 함수의 다중 정의(overloading) (1)

- 함수의 이름은 같고 전달하는 인수들이 다른 경우
- 함수 호출에서 보면 하나의 이름으로 여러 가지 기능을 할 수 있는 다형성을 제공
- 프로그래밍 작업에서 보면 다형성이 있는 코드를 일일이 코딩해야 하는 번거롭고 비효율적인 작업으로 코드의 재활용과는 먼 개념

## ■ 함수의 다중 정의(overloading) (2)

[예시]

```
int Add(int x, int y){
    int m = x + y;
    return m;
}

double Add(double x, double y){
    double m = x + y;
    return m;
}

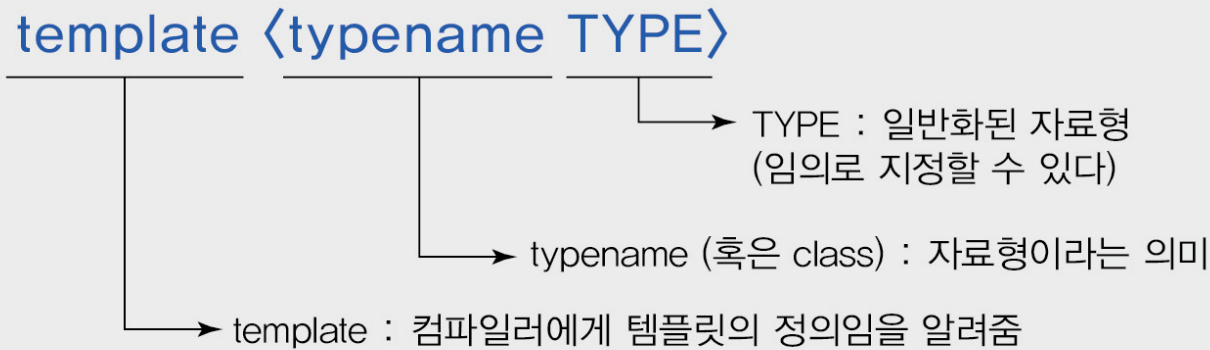
void main(){
    Add(3, 8);           // Add(int x, int y) 함수를 호출한다.
    Add(3.5, 8.4);       // Add(double x, double y) 함수를 호출한다.
}
```

## ■ 템플릿

- 템플릿(template)은 객체지향 프로그래밍의 특징 중 하나인 코드 재활용을 제공
- 동일한 알고리즘에서 자료형이 다를 때 여러 번 코딩하지 않고 일반형으로 한번만 코딩하여 다양한 자료형을 연결하여 처리해 줌
- 다형성이 있는 함수나 클래스를 C++ 컴파일러가 제공해 주므로 함수를 일반화할 수 있음
- 일반화 프로그래밍(generic programming)
  - 템플릿은 구체적인 자료형 대신에 **일반형(generic type)**으로 프로그래밍하는 것임

# 템플릿 함수

## ■ 템플릿 선언



## ■ 템플릿 함수의 형식

```
template <typename 템플릿_자료형_이름>  
TYPE 함수_이름  
{  
    .....  
}
```

TYPE

# 템플릿 함수

## ■ 템플릿 함수

[예시]

```
template <typename T>
```

```
T Add(T x, T y){
```

```
    T m = x + y;
```

```
    return m;
```

```
}
```

```
void main(){
```

```
    Add(3, 8);
```

```
    Add(3.5, 8.4);
```

```
}
```



## ■ 템플릿 함수의 호출

```
template <typename T>
T Add(T x, T y)
{
    T m = x + y;
    return m;
}
```

Add(3, 8)으로 호출하는 경우

```
int Add(int x, int y)
{
    int m = x + y;
    return m;
}
```

Add(3.5, 8.4)으로 호출하는 경우

```
double Add(double x, double y)
{
    double m = x + y;
    return m;
}
```

## ■ ex12\_1.cpp (Add() 함수를 템플릿으로 구현)

```
#include <iostream>
using namespace std;
template <typename T>
```

```
T Add(T x, T y)
```

```
{
    T m = x + y;
    return m;
}
```

```
void main()
```

```
{
    int a = 3, b = 8, c;
    double d = 3.5, e = 8.9, f;

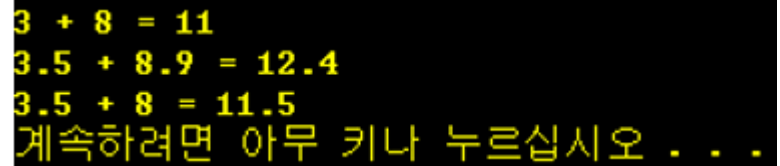
    c = Add(a, b);
    cout << a << " + " << b << " = " << c << endl;
    f = Add(d, e);
    cout << d << " + " << e << " = " << f << endl;
}
```

```
3 + 8 = 11
3.5 + 8.9 = 12.4
계속하려면 아무 키나 누르십시오 . . .
```

## ■ ex12\_2.cpp (자료형이 두 개인 경우)

```
#include <iostream>
using namespace std;
template <typename TYPE1, typename TYPE2>
TYPE1 Add(TYPE1 x, TYPE2 y)
{
    TYPE1 m = x + y;
    return m;
}
void main()
{
    int a = 3, b = 8, c;
    double d = 3.5, e = 8.9, f;

    c = Add(a, b);
    cout << a << " + " << b << " = " << c << endl;
    f = Add(d, e);
    cout << d << " + " << e << " = " << f << endl;
    f = Add(d, b);
    cout << d << " + " << b << " = " << f << endl;
}
```



```
3 + 8 = 11
3.5 + 8.9 = 12.4
3.5 + 8 = 11.5
계속하려면 아무 키나 누르십시오 . . .
```

# 템플릿 클래스

## ■ 템플릿 클래스

- 일반화된 클래스를 정의하여 사용
- 템플릿 클래스를 정의
  - 먼저 '**template <typename TYPE>**' 으로 템플릿 선언을 하고
  - 클래스를 정의하는 문장 내에서 해당 자료형을 **템플릿 자료형(TYPE)** 으로 지정

```
template <typename 템플릿_자료형_이름>  
class 클래스_이름 {  
    템플릿_자료형_이름 변수명;  
  
    .....  
  
};
```

# 템플릿 클래스

## ■ 템플릿 클래스

- 클래스를 정의할 때뿐만 아니라 **멤버함수를 정의할 때에도 'template <typename TYPE>'으로 템플릿을 선언**하여야 함
- 멤버함수를 정의할 때 **스코프 연산자(::)** 앞의 **클래스 이름 다음에도 템플릿 자료형(TYPE)**을 붙여야 함

```
template <typename 템플릿_자료형_이름>  
반환형 클래스_이름 <템플릿_자료형_이름>::멤버_함수  
{  
    .....  
};
```

# 템플릿 클래스

## ■ 템플릿 클래스 예시

```
template <typename TYPE>
```

```
// 템플릿 선언
```

```
class Stack
```

```
{
```

```
protected:
```

```
    TYPE *pstack;
```

```
    int stsize;
```

```
public:
```

```
    void push(TYPE value);
```

```
    TYPE pop();
```

```
    ...
```

```
};
```

```
template <typename TYPE>
```

```
// 템플릿 선언
```

```
void Stack<TYPE>::push(TYPE data)
```

```
{
```

```
    ...
```

```
}
```

```
template <typename TYPE>
```

```
// 템플릿 선언
```

```
TYPE Stack<TYPE>::pop()
```

```
{
```

```
    ...
```

```
}
```

## ■ 템플릿 클래스

- 템플릿은 프로그램이 실행 시간(run time)에 클래스나 함수를 만들지 않고 소스 코드를 컴파일 시간(compile time)에 클래스나 함수를 만듦
  - 템플릿을 사용하더라도 프로그램 실행이 느려 지지 않음
- 클래스 선언과 함께 템플릿 함수의 정의를 헤더파일에 함께 두어야 함
  - 일반적인 경우 중복 정의를 방지하기 위하여 선언은 헤더 파일(\*.h)에, 정의는 구현 파일(\*.cpp)에 둠
  - 그러나 템플릿 함수는 함수 만드는 방법을 가르쳐주는 것이므로 반드시 클래스 선언과 함께 템플릿 함수의 정의를 헤더 파일에 함께 두어야 함

## ■ 예제 12.3 stack1.h (1) (템플릿 클래스 Stack의 구현 예)

```
template <typename T>
```

```
class Stack {
```

```
protected:
```

```
    T* pstack;        // 할당된 메모리 주소
```

```
    int stsize;        // Stack의 크기
```

```
    int top;           // Stack의 top
```

```
public:
```

```
    Stack(int size = 10);
```

```
    ~Stack();
```

```
    void push(T value, int &pt);
```

```
    T pop(int &pt);
```

```
};
```

```
template <typename T> Stack<T>::Stack(int size) {
```

```
    stsize = size;
```

```
    pstack = new T[stsize];
```

```
    top = 0;
```

```
}
```

```
template <typename T> Stack<T>::~~Stack() {
```

```
    delete [] pstack;
```

```
    stsize = 0;
```

```
}
```



## ■ 예제 12.3 stack1.h (2) (템플릿 클래스 Stack의 구현 예)

```
template <typename T> void Stack<T>::push(T value, int &pt) {
    if (top < stsize) {
        pt = top;
        pstack[top++] = value;
    }
    else
        cout << "스택이 가득 차 있습니다." << endl;
}

template <typename T> T Stack<T>::pop(int &pt) {
    if (top > 0) {
        T value = pstack[--top];
        pt = top;
        return value;
    }
    else {
        cout << "스택이 비어 있습니다." << endl;
        pt = -1;
        return 0;
    }
}
```

## ■ 예제 12.3 ex12\_3.cpp (1) (템플릿 클래스 Stack의 구현 예)

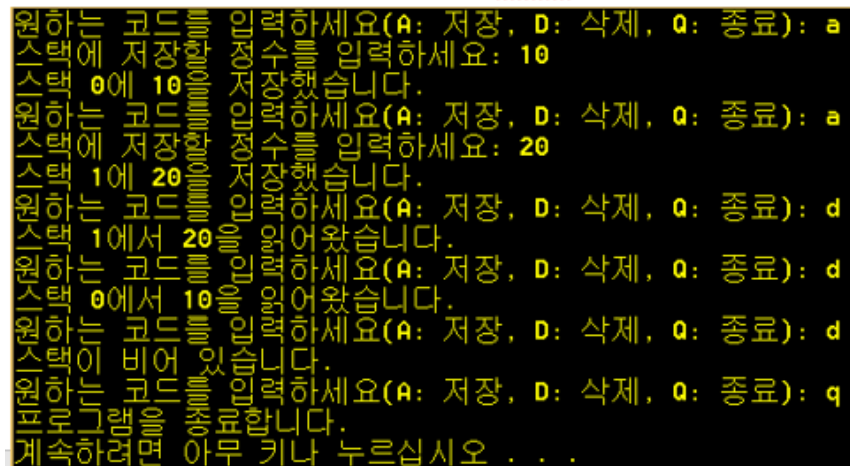
```
#include <iostream>
using namespace std;
#include "stack1.h"
int main() {
    Stack<int> st(5);
    int pt, value;  char code;
    cout << "원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): ";
    cin >> code;
    while (code != 'Q' && code != 'q') {
        while (cin.get() != '\n')
            continue;
        switch (code) {
        case 'A':
        case 'a':
            cout << "스택에 저장할 정수를 입력하세요: ";
            cin >> value;
            st.push(value, pt);
            cout << "스택 " << pt << "에 " << value << "을 저장했습니다." << endl;
            break;
```

# 템플릿 클래스

## ■ 예제 12.3 ex12\_3.cpp (2) (템플릿 클래스 Stack의 구현 예)

```
case 'D':
case 'd':
    value = st.pop(pt);
    if (pt < 0) break;
    cout << "스택 " << pt << "에서 " << value << "을 읽어왔습니다." << endl;
    break;
}
cout << "원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): ";
cin >> code;
}
cout << "프로그램을 종료합니다." << endl;
return 0;
```

```
}
```



```
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): a
스택에 저장한 인정수를 입력하세요: 10
0에 10을 저장했습니다.
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): a
스택에 저장한 인정수를 입력하세요: 20
1에 20을 저장했습니다.
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): d
스택 1에서 20을 읽어왔습니다.
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): d
스택 0에서 10을 읽어왔습니다.
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): d
스택이 비어 있습니다.
원하는 코드를 입력하세요(A: 저장, D: 삭제, Q: 종료): q
프로그램을 종료합니다.
계속하려면 아무 키나 누르십시오 . . .
```

## ■ 예제 12.4 cal.h (둘 이상의 자료형의 템플릿 클래스)

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
class Cal {
private:
    T1 data1,
    T2 data2;
public:
    Cal() { };
    void Add(T1 d1, T2 d2);
    void Mul(T1 d1, T2 d2) ;
};
template <typename T1, typename T2> void Cal<T1, T2>::Add(T1 d1, T2 d2) {
    T1 d3;
    d3 = d1 + d2;
    cout << d1 << " + " << d2 << " = " << d3 << endl;
}
template <typename T1, typename T2> void Cal<T1, T2>::Mul(T1 d1, T2 d2) {
    T1 d3;
    d3 = d1 * d2;
    cout << d1 << " * " << d2 << " = " << d3 << endl;
}
```

# 템플릿 클래스

## ■ 예제 12.4 ex12\_4.cpp (둘 이상의 자료형의 템플릿 클래스)

```
#include "cal.h"
void main()
{
    Cal<double, double> obj;
    double a = 15.5;
    int b = 5;

    obj.Add(a, b);
    obj.Mul(a, b);
    obj.Add(10.2, 20.5);
    obj.Mul(20, 10);
}
```

```
15.5 + 5 = 20.5
15.5 * 5 = 77.5
10.2 + 20.5 = 30.7
20 * 10 = 200
계속하려면 아무 키나 누르십시오 . . .
```

# C++ 스타일의 문자열

## ■ C++ 스타일의 문자열

- C++ 스타일의 문자열은 `string`이라는 클래스를 사용
- `string` 클래스는 `basic_string<char>` 템플릿 클래스를 `typedef`를 사용하여 재정의한 것

## ■ C 스타일의 문자열

- C에서는 문자의 배열을 사용하여 문자열을 처리
- 문자열의 길이가 배열의 크기보다 길면 여러 가지 예기치 않은 문제가 발생하기 때문에 배열의 크기에 주의
- 또한 문자열의 모든 처리는 문자열 함수를 사용하여야 하는 불편함이 있음
- C 스타일의 문자열 처리 함수
  - C에서는 `string.h`를 통하여 지원
  - C++에서는 `cstring`을 통하여 지원

# C++ 스타일의 문자열

## ■ string 클래스의 객체

- C++ 스타일의 문자열은 string이라는 클래스의 객체를 사용
  - string 클래스는 문자 배열의 멤버 변수와 문자열 처리를 위한 많은 멤버함수들을 제공하고 연산자가 다중 정의되어 있기 때문에 편하고 안전하게 문자열을 사용할 수 있음
- string 클래스는 string 헤더 파일에 정의되어 있고 std 이름 공간에 속해 있음
  - string 클래스를 사용하려면 string 헤더파일을 포함시켜야 하고 std 이름 공간을 사용

## ■ String 객체의 선언

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

string 문자열_객체_이름;
string 문자열_객체_이름 = "초기값";
string 문자열_객체_이름("초기값");
```

[예시]

```
string str;                // string 객체 str을 생성
string str("Korea.");      // string 객체 str을 생성하고 초기화
```

# C++ 스타일의 문자열

## ■ string 클래스의 생성자

멤버 함수	설 명
string()	기본 생성자, 크기가 0인 string 객체를 생성한다.
string(const char *s)	문자열 s로 초기화된 string 객체를 생성한다.
string(const char *s, int n)	문자열 s로 초기화된 string 객체를 생성한다. n개의 문자까지만 초기화한다.
string(int n, char c)	문자 c로 초기화된 n개의 문자로 이루어진 string 객체를 생성한다.
string(const string &src)	복사 생성자, string 객체 src를 복사하여 string 객체를 생성한다.



# C++ 스타일의 문자열

## ■ string 클래스의 멤버 함수

멤버 함수	설 명
empty()	문자열이 비어있는지 검사한다. 비어있으면 true 반환
insert(pos, s)	pos 위치에 문자열 s를 삽입
append(s)	끝에 문자열 s를 덧붙임
remove(pos, len)	pos 위치부터 len 길이만큼을 삭제
find(pos, s)	pos 위치부터 문자열 s가 발견되는 첫 번째 색인을 반환
find(s)	처음부터 문자열 s가 발견되는 첫 번째 색인을 반환
length()	문자열의 문자수를 반환(NULL 문자는 제외)

# C++ 스타일의 문자열


## ■ ex12\_5.cpp (1) (string 객체의 사용)

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
    char str[] = "01234567890123456789";
    string s1;                                     // 기본 생성자 호출

    cout << "문자열을 입력 하시오: ";
    cin >> s1;
    cout << "s1 : " << s1 << endl;

    string s2("C++ Programming");
    cout << "s2 : " << s2 << endl;

    string s3(str, 10);
    cout << "s3 : " << s3 << endl;
    cout << "s3의 길이 : " << s3.length() << endl;
```



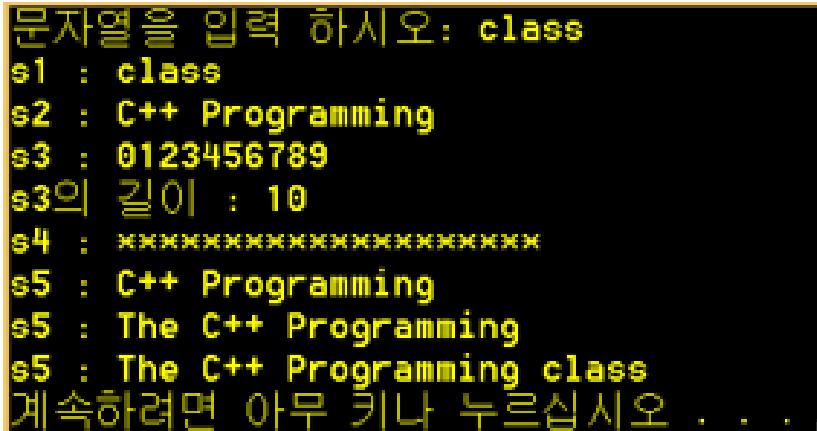
```
문자열을 입력 하시오: class
s1 : class
s2 : C++ Programming
s3 : 0123456789
s3의 길이 : 10
```

# C++ 스타일의 문자열

## ■ ex12\_5.cpp (2) (string 객체의 사용)

```
string s4(20, '*');  
cout << "s4 : " << s4 << endl;  
string s5(s2);  
cout << "s5 : " << s5 << endl;  
s5.insert(0, "The ");  
cout << "s5 : " << s5 << endl;  
s5.append(" ");  
s5.append(s1);  
cout << "s5 : " << s5 << endl;  
}
```

// 복사 생성자 호출



```
문자열을 입력 하시오: class  
s1 : class  
s2 : C++ Programming  
s3 : 0123456789  
s3의 길이 : 10  
s4 : xxxxxxxxxxxxxxxxxxxxxxxx  
s5 : C++ Programming  
s5 : The C++ Programming  
s5 : The C++ Programming class  
계속하려면 아무 키나 누르십시오 . . .
```

## ■ ex12\_6.cpp (string 객체의 사용)

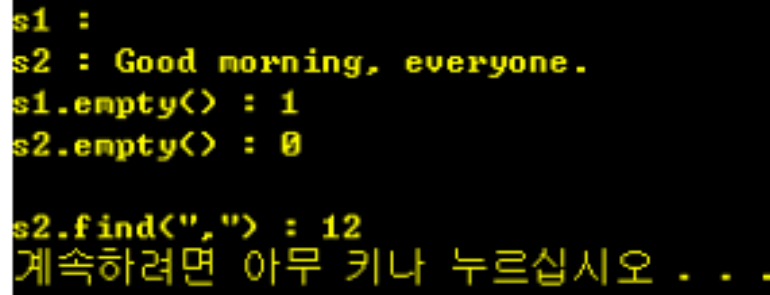
```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s1;
    string s2 = "Good morning, everyone." ;

    cout << "s1 : " << s1 << endl;
    cout << "s2 : " << s2 << endl;

    cout << "s1.empty() : " << s1.empty() << endl;
    cout << "s2.empty() : " << s2.empty() << endl << endl;

    cout << "s2.find(' ','') : " << s2.find(",") << endl;

    return 0;
}
```



```
s1 :
s2 : Good morning, everyone.
s1.empty() : 1
s2.empty() : 0

s2.find(",") : 12
계속하려면 아무 키나 누르십시오 . . .
```

## ■ string 객체의 입력

### ■ >> 연산자 사용

### ■ getline() 함수 사용

- getline() 함수는 string 클래스에서 사용할 수 있지만 cin의 멤버함수는 아니므로 getline(cin, str) 으로 입력 받아야 함
- 공백 문자도 입력 가능
- \n 문자는 읽어서 버림

[예시]

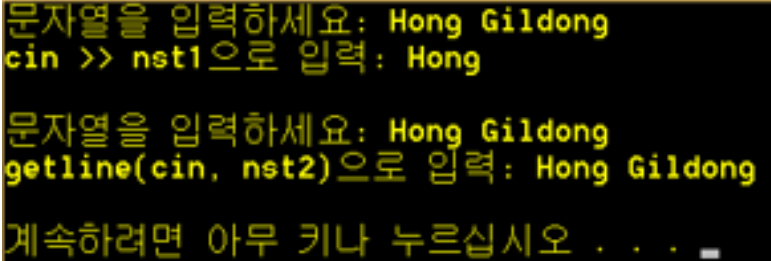
```
string str;           // string 객체 str을 생성
cin >> str;           // 한 단어만 읽을 수 있다. 빈 칸은 읽어들이지 않는다.
getline(cin, str);     // 한 줄을 읽을 수 있다. \n은 읽어서 버린다.
```

## ■ ex12\_7.cpp (string 객체의 입력)

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
    string nst1, nst2;

    cout << "문자열을 입력하세요: ";
    cin >> nst1;
    cout << "cin >> nst1)으로 입력: " << nst1 << endl << endl;
    fflush(stdin);

    cout << "문자열을 입력하세요: ";
    getline(cin, nst1);
    cout << "getline(cin, nst2)으로 입력: " << nst2 << endl << endl;
}
```



A terminal window showing the execution of the C++ program. The first prompt is "문자열을 입력하세요: Hong Gildong", followed by the output "cin >> nst1)으로 입력: Hong". The second prompt is "문자열을 입력하세요: Hong Gildong", followed by the output "getline(cin, nst2)으로 입력: Hong Gildong". The final line shows the prompt "계속하려면 아무 키나 누르십시오 . . . \_".

## ■ string 클래스의 연산자

- string 클래스는 연산자들이 다중 정의되어 있어서 이를 사용하여 문자열을 쉽게 처리할 수 있음
- + 연산자 : 문자열 클래스 객체를 더할 때, 즉 문자열을 연결
- = 연산자 : 어떤 문자열 객체를 다른 문자열 객체에 대입
- += 연산자 : 기존의 문자열에 다른 문자열을 덧붙임
- == 연산자 : 두 개의 문자열이 동일한지를 검사할 수 있음
- < 연산자, > 연산자 : 문자열의 사전에서의 순서를 검사
- [ ] 연산자 : 배열처럼 색인으로 하나의 문자를 끄집어 낼 수 있음
- >> 와 << 연산자 : 입출력 할 수 있음
  - >> 연산자로 공백 문자는 입력할 수 없음
  - getline() 함수를 사용하여야 함

## ■ ex12\_8.cpp (1) (string 클래스의 연산자)

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s1 = "Good" ;
    string s2 = " morning." ;
    string s3 = "Good" ;
    string s4;
    cout << "s1 : " << s1 << endl;
    cout << "s2 : " << s2 << endl;
    cout << "s3 : " << s3 << endl;
    cout << "s4 : " << s4 << endl << endl;
    s4 = s1;
    cout << "(s4 = s1) --> s4 : " << s4 << endl;
    s4 += s2;
    cout << "(s4 += s2) --> s4 : " << s4 << endl;
    cout << "(s1 == s2) --> " << (s1 == s2) << endl;
    cout << "(s1 == s3) --> " << (s1 == s3) << endl;
    return 0;
}
```

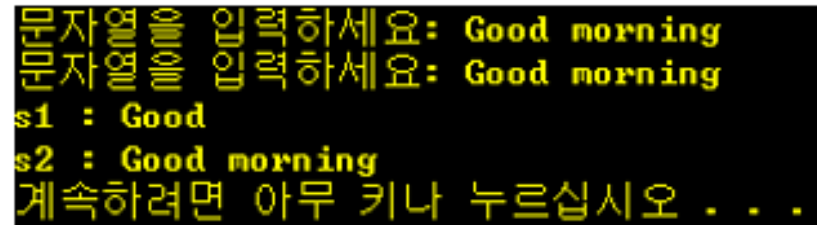
```
s1 : Good
s2 :  morning.
s3 : Good
s4 :

(s4 = s1) --> s4 : Good
(s4 += s2) --> s4 : Good morning.
(s1 == s2) --> 0
(s1 == s3) --> 1
계속하려면 아무 키나 누르십시오 . . .
```



## ■ ex12\_9.cpp (1) (string 클래스의 연산자)

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    int i = 0;
    string s1, s2;
    cout << "문자열을 입력 하세요: " ;           cin >> s1;
    fflush(stdin);
    cout << "문자열을 입력 하세요: " ;           getline(cin, s2);
    cout << "s1 : " << s1 << endl;
    cout << "s2 : " ;
    while (s2[i] != NULL) {
        cout << s2[i] ;
        i++;
    }
    cout << endl;
    return 0;
}
```



문자열을 입력하세요: Good morning  
문자열을 입력하세요: Good morning  
s1 : Good  
s2 : Good morning  
계속하려면 아무 키나 누르십시오 . . .

# 표준 템플릿 라이브러리

## ■ 표준 템플릿 라이브러리(STL, standard template library)

- 프로그래머들이 많이 사용하는 클래스와 함수들을 템플릿을 사용하여 만들어 둔 클래스
- 템플릿을 사용하였으므로 어떠한 자료형에도 적용할 수 있음
- STL은 일반화 프로그래밍(generic programming)의 예임
- STL의 클래스는 이미 검증을 마쳤기 때문에 안심하고 사용할 수 있음
- STL을 사용하여 프로그램을 만들면 오류 없는 프로그램을 빠른 시일 내에 쉽게 개발할 수 있음
- STL을 이용하여 프로그램을 쉽게 개발하기 위하여 STL에는 **컨테이너, 반복자, 알고리즘의 세 가지 구성 요소를 제공**

## ■ STL의 요소

### ■ 컨테이너(container)

- 자료를 저장하는 창고와 같은 클래스
- 컨테이너는 자료를 저장하는 방식과 자료들을 처리하는 방식에 따라 **순차 컨테이너 (sequential container)**, **연관 컨테이너(associative container)** 그리고 **어댑터 컨테이너 (adapter container)**로 나뉨

### ■ 반복자(iterator)

- 컨테이너에 순서대로 저장된 자료들에 대하여 필요한 처리를 하기 위하여 **자료들을 첫 번째 위치부터 마지막 위치까지 순서대로 가리키는 역할**을 함

### ■ 알고리즘

- **자료들을 적당히 가공하고 사용할 수 있는 방법**
- STL에서는 알고리즘을 컨테이너와는 별개로 작성되어 있어 컨테이너 종류에 관계없이 사용할 수 있도록 되어 있음

# 표준 템플릿 라이브러리

## ■ 순차 컨테이너

- 자료를 순차적으로 저장
- 벡터(vector), 리스트(list)과 같은 것들이 있음

## ■ vector 템플릿 클래스

- vector는 vector 템플릿 클래스로 정의되어 있음
- 자동으로 배열의 크기 조절과 객체의 추가, 삭제가 가능한 동적 배열 구조를 가짐
- vector 템플릿 클래스를 사용하려면 **vector 헤더 파일을 포함**하여야 함

## ■ vector 템플릿 클래스의 사용

```
vector<int> score1(4);
```

```
// int 성적을 저장할 score1 이라는 이름의
```

```
// vector 컨테이너를 생성한다. 원소의 개수는 4개이다.
```

```
cin >> score[1];
```

```
// 콘솔에서 입력받아 score의 두 번째에 원소에 저장
```

```
cout << score[1];
```

```
// score의 두 번째 원소 값을 콘솔에 출력
```

```
int arr[] = {10, 20, 30, 40, 50, 60, 70};
```

```
vector<int> score2(arr, arr+4);
```

```
// 배열 arr에서 4개의 요소를 가지는 vector 컨테이너 score2를
```

```
// 생성한다. arr은 시작 반복자의 위치이고, arr+4은 마지막
```

```
// 요소의 하나 다음의 위치로 제외된다. 따라서, score2의
```

```
// 첫 번째 요소는 배열 arr의 첫 번째 요소(10)이고,
```

```
// 마지막 요소는 arr+4의 하나 전까지인 40으로 초기화 된다.
```

## ■ vector 템플릿 클래스의 사용

[예시]

```
vector<int> score(4);    // 정수(int)를 저장할 vector 컨테이너  
                        // score 생성, 원소의 개수는 4이다.
```

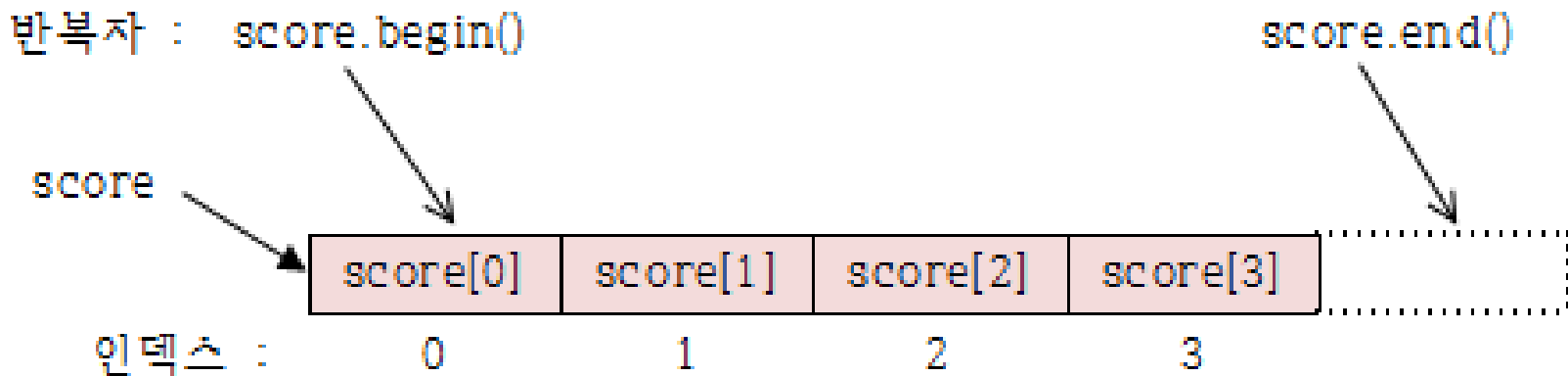
### ■ 멤버 함수

- size() 함수 : 컨테이너의 크기를 구함
- push\_back(value) 함수 : 컨테이너의 끝에 value라는 데이터를 추가
- pop\_back() 함수 : 컨테이너의 마지막 요소를 삭제
- front() 함수 : 컨테이너의 첫 번째 요소를 반환
- insert(iterator, value) 함수 : 컨테이너의 중간(반복자의 위치)에 value를 삽입
- begin() 함수 : 첫 번째 요소의 반복자 위치를 구함
- end() 함수 : 마지막 요소를 지난 반복자의 위치를 구함

# 표준 템플릿 라이브러리

## ■ vector 템플릿 클래스의 사용

- Vector 컨테이너와 반복자

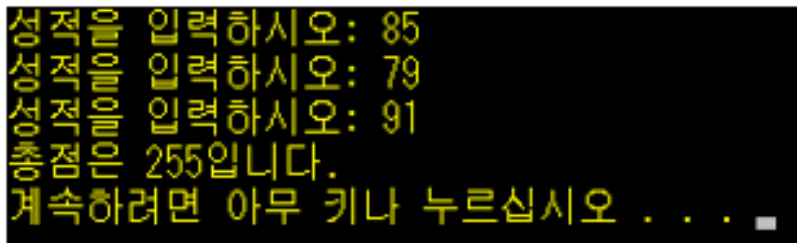


## ■ ex12\_10.cpp (vector 템플릿 클래스의 사용)

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

void main()
{
    unsigned int i;
    int sum;
    vector<int> score(3);

    for (i=0; i<score.size(); i++) {
        cout << "성적을 입력하시오: ";
        cin >> score[i];
    }
    for (i=0, sum=0; i<score.size(); i++)
        sum += score[i];
    cout << "총점은 " << sum << "입니다." << endl;
}
```



```
성적을 입력하시오: 85
성적을 입력하시오: 79
성적을 입력하시오: 91
총점은 255입니다.
계속하려면 아무 키나 누르십시오 . . .
```



## ■ ex12\_11.cpp (1) (vector 템플릿 클래스의 사용)

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Student {
private :
    int id;
    string name;
public :
    Student(int i = 0, string n = "") {
        id = i, name = n;
    }
    void show() {
        cout << id << " : " << name << endl;
    }
};
void main()
{
    vector<Student> classA;
    classA.push_back(Student(1501, "Kim" ));
    classA.push_back(Student(1505, "Lee" ));
    classA.push_back(Student(1503, "Park" ));
```

# 표준 템플릿 라이브러리

## ■ ex12\_11.cpp (2) (vector 템플릿 클래스의 사용)

```
vector<Student>::iterator it;
```

```
for (it = classA.begin() ; it != classA.end() ; it++)  
    it->show();  
cout << endl;
```

```
classA.pop_back();  
for (it = classA.begin() ; it != classA.end() ; it++)  
    it->show();  
cout << endl;
```

```
classA.insert(classA.begin()+1, Student(1502, "Hong"));  
for (it = classA.begin(); it != classA.end(); it++)  
    it->show();
```

```
}
```

```
1501 : Kim  
1505 : Lee  
1503 : Park
```

```
1501 : Kim  
1505 : Lee
```

```
1501 : Kim  
1502 : Hong  
1505 : Lee
```

```
계속하려면 아무 키나 누르십시오 . . .
```

# 표준 템플릿 라이브러리

## ■ 연관 컨테이너

- 연관 컨테이너는 원소들을 검색하기 위하여 키(key)를 가지고 있는 사전과 같은 구조를 사용
- 집합(set), 맵(map)과 같은 것들이 있음

## ■ map 템플릿 클래스

- 원하는 데이터를 빠르게 찾을 수 있는 자료구조로 사전과 같이 키(key)가 있고 이 키에 연관된 값(value)이 있는 구조
- **맵에 자료를 추가하고 검색하는 방법**으로는 인덱스 연산자 [ ]를 사용하는 방법이 편리함
- map 템플릿 클래스를 사용하려면 **map 헤더 파일을 포함**하여야 함

## ■ map 템플릿 클래스

[예시]

- string 자료의 key와 string 자료의 value를 가지고 있는 animal 이라는 map 컨테이너는 다음과 같이 정의할 수 있음

```
map<string, string> animal;
```

```
// string 자료의 key 값과 string 자료의 value를
```

```
// 가지고 있는 animal 이라는 map 컨테이너를 정의
```

## ■ ex12\_12.cpp (1) (map 템플릿 클래스의 사용)

```
#include <iostream>
```

```
#include <string>
```

```
#include <map>
```

```
using namespace std;
```

```
void main()
```

```
{
```

```
    map<string, string> animal;
```

```
    string word;
```

```
    animal["deer"] = "사슴";
```

```
    animal["horse"] = "말";
```

```
    animal["cat"] = "고양이";
```

```
    animal["dog"] = "개";
```

```
    animal["tiger"] = "호랑이";
```

```
    map<string, string> :: iterator it;
```

```
    for (it = animal.begin(); it != animal.end(); it++)
```

```
        cout << it->first << " : " << it->second << endl;
```

```
    cout << endl;
```

# 표준 템플릿 라이브러리

## ■ ex12\_12.cpp (2) (map 템플릿 클래스의 사용)

```
while (1) {  
    cout << "영어 단어를 입력 하세요 (끝내려면 Ctrl-Z) : " ;  
    cin >> word ;  
    if (cin.eof())  
        break;           // Ctrl-Z이면 끝내기  
    cout << word << " : " << animal[word] << endl;  
}  
}
```

```
cat : 고양이  
deer : 사슴  
dog : 개  
horse : 말  
tiger : 호랑이  
  
영어 단어를 입력 하세요 (끝내려면 Ctrl-Z) : dog  
dog : 개  
영어 단어를 입력 하세요 (끝내려면 Ctrl-Z) : lion  
lion :  
영어 단어를 입력 하세요 (끝내려면 Ctrl-Z) : ^Z  
계속하려면 아무 키나 누르십시오 . . .
```

# 표준 템플릿 라이브러리

## ■ STL 알고리즘

- STL 알고리즘은 자료를 처리하기 위한 탐색, 정렬, 삽입, 삭제 등과 같은 프로그램 구현에 많이 사용되는 기능을 함수 템플릿으로 제공하는 범용 함수
- STL의 알고리즘을 사용하려면 **#include <algorithm>을 사용**하여야 함

### ■ sort(begin, end) 함수

- sort(begin, end) 함수는 범위 내의 값을 오름차순으로 정렬하는 함수
- 첫 번째 파라미터(begin)는 정렬을 시작하는 반복자의 위치이고,
- 두 번째 파라미터(end)는 마지막 요소의 하나 다음의 위치로 end는 제외

### ■ find(begin, end, value) 함수

- STL의 find(begin, end, value) 함수는 범위에서 지정된 값을 가진 요소가 첫 번째로 나타나는 위치를 찾음
  - 첫 번째 파라미터(begin)는 지정된 값을 검색할 범위의 첫 번째 요소의 위치 주소를 지정하는 입력 반복자이고,
  - 두 번째 파라미터(end)는 지정된 값을 검색할 범위의 마지막 요소 하나 다음의 위치 주소를 지정하는 입력 반복자로 end는 제외
  - 세 번째 파라미터(value)는 검색할 값

## ■ ex12\_13.cpp (1) (sort() 알고리즘의 사용)

```
#include <iostream>
#include <algorithm>
using namespace std;
void main()
{
    int a[4] = {50, 10, 20, 40};

    cout << "정렬하지 않았을 때 : " ;
    for (int i=0; i<4; i++)
        cout << a[i] << " " ;
    cout << endl;

    sort(a, a+4);

    cout << "정렬했을 때 : " ;
    for (int i=0; i<4; i++)
        cout << a[i] << " " ;
    cout << endl;
}
```

```
정렬하지 않았을 때 : 50 10 20 40
정렬했을 때 : 10 20 40 50
계속하려면 아무 키나 누르십시오 . . .
```



## ■ ex12\_14.cpp (1) (STL의 사용)

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
void main()
{
    int arr[4] = {10, 20, 30, 15};
    int value;
    vector<int> score(&arr[0], &arr[4]);
    vector<int> :: iterator it;

    cout << "정렬하지 않았을 때 : ";
    for (it = score.begin(); it < score.end(); it++)
        cout << *it << " ";
    cout << endl;

    cout << "정렬했을 때 : ";
    sort(score.begin(), score.end());
    for (it = score.begin(); it < score.end(); it++)
        cout << *it << " ";
    cout << endl;
```

# 표준 템플릿 라이브러리

## ■ ex12\_14.cpp (2) (STL의 사용)

```
cout << "찾을 값을 입력 하시오 : ";  
cin >> value;
```

```
it = find(score.begin(), score.end(), value);
```

```
if (it != score.end())
```

```
    cout << value << "는 " << it - score.begin() + 1 << "번째에 있습니다." << endl;
```

```
else
```

```
    cout << value << "는 없습니다." << endl;
```

```
}
```

```
정렬하지 않았을 때 : 10 20 30 15  
정렬했을 때 : 10 15 20 30  
찾을 값을 입력 하시오 : 15  
15는 2번째에 있습니다.  
계속하려면 아무 키나 누르십시오 . . .
```

```
정렬하지 않았을 때 : 10 20 30 15  
정렬했을 때 : 10 15 20 30  
찾을 값을 입력 하시오 : 25  
25는 없습니다.  
계속하려면 아무 키나 누르십시오 . . .
```



Thank You

---