

12주차 정리노트

note#7

2023. 11. 20(월)
16조 이수영, 원준서

목차

실습(ppt p3-p22)

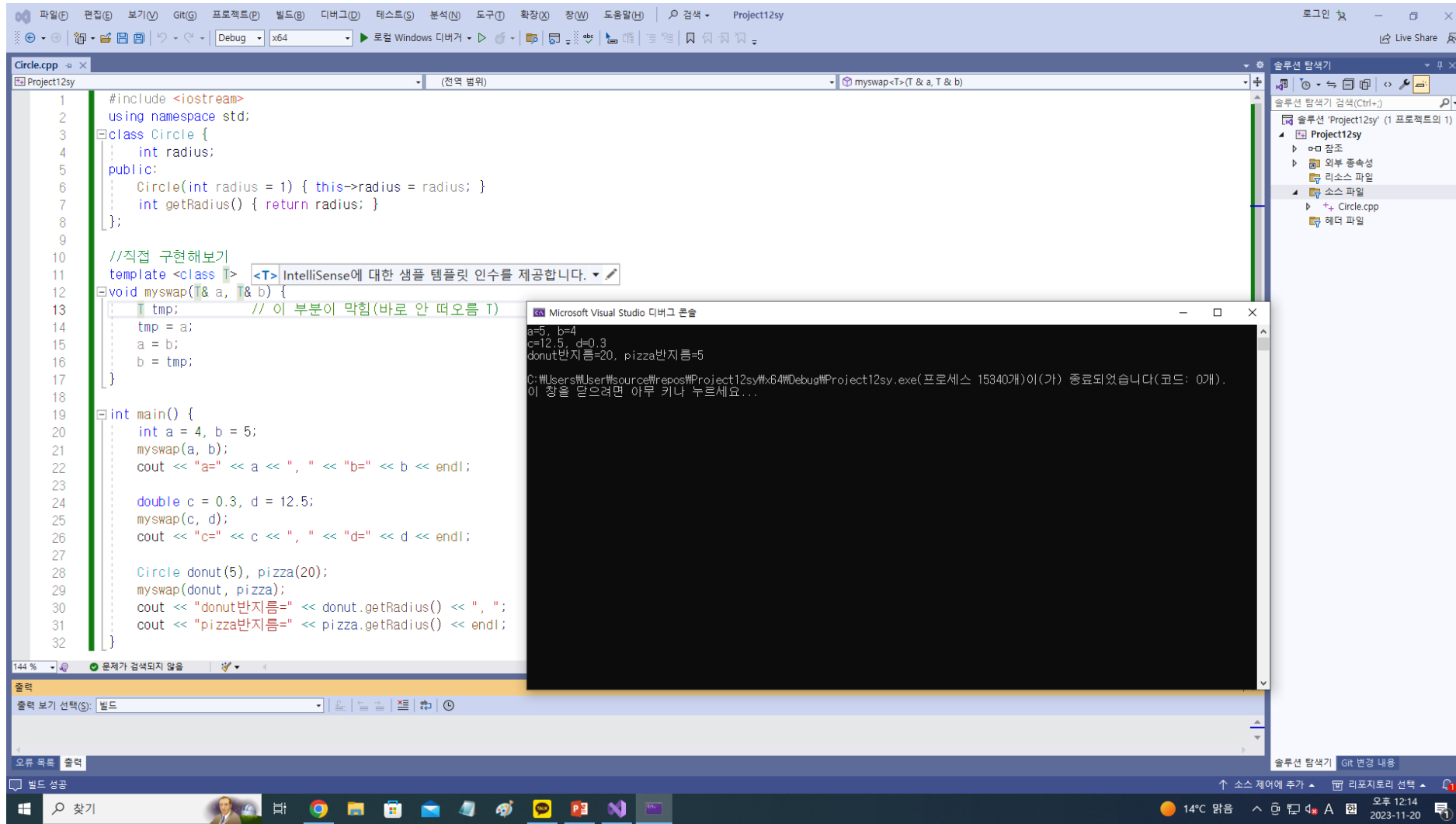
- [1차시] 예제1 T 구현
- [1차시] 예제2 T 구현
- [1차시] 예제3 (1) int 구현
- [1차시] 예제3 (2) T 구현
- [2차시] 예제6 T '클래스'
- [2차시] 예제8 2개이상 제네릭타입
- [3차시] p.21+p.24 vector API
- [3차시] 예제9 vector 컨테이너
- [3차시] 예제11 **iteraotor 사용** vector

정리노트(ppt p23-p42)

- 일반화와 템플릿
- 템플릿으로부터의 구체화
- 템플릿 장점과 제네릭 프로그래밍
- C++ 표준 템플릿 라이브러리 STL
- STL과 관련된 헤더 파일과 이름 공간
- (1) vector 컨테이너
- iterator 사용
- (2) map 컨테이너
- STL 알고리즘 사용하기
- C++에서의 auto

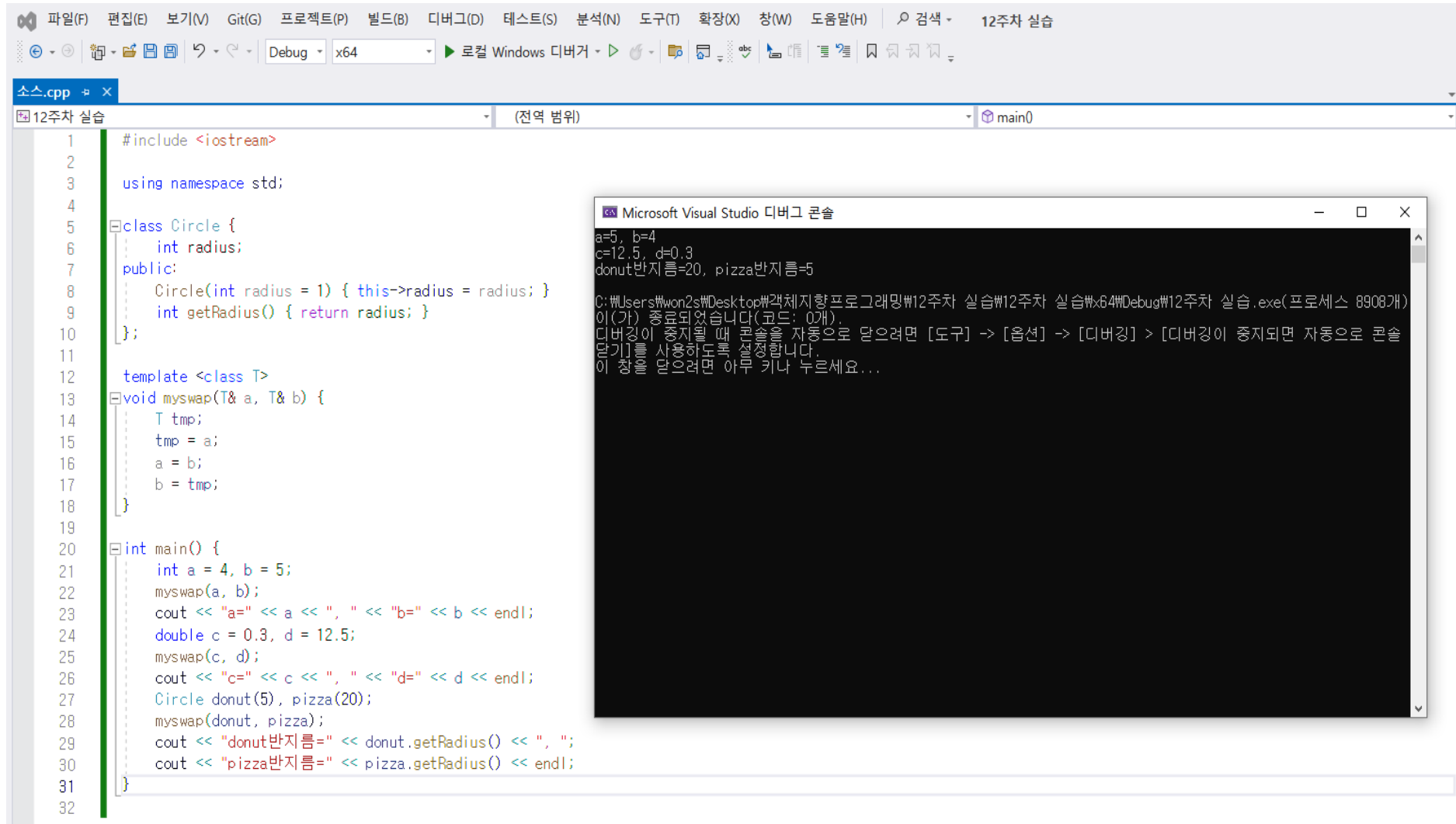
[1차시 이수영] 예제 10-1

template 안 보고 직접 구현



[1차시 원준서] 예제 10-1

template 안 보고 직접 구현



The image shows a Visual Studio IDE window with a C++ source file named '소스.cpp'. The code defines a `Circle` class and a `myswap` template function. The `main` function tests these with various data types and a `Circle` object. A debug console window is open, showing the output of the program.

```
#include <iostream>

using namespace std;

class Circle {
    int radius;
public:
    Circle(int radius = 1) { this->radius = radius; }
    int getRadius() { return radius; }
};

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

int main() {
    int a = 4, b = 5;
    myswap(a, b);
    cout << "a=" << a << ", " << "b=" << b << endl;
    double c = 0.3, d = 12.5;
    myswap(c, d);
    cout << "c=" << c << ", " << "d=" << d << endl;
    Circle donut(5), pizza(20);
    myswap(donut, pizza);
    cout << "donut반지름=" << donut.getRadius() << ", ";
    cout << "pizza반지름=" << pizza.getRadius() << endl;
}
```

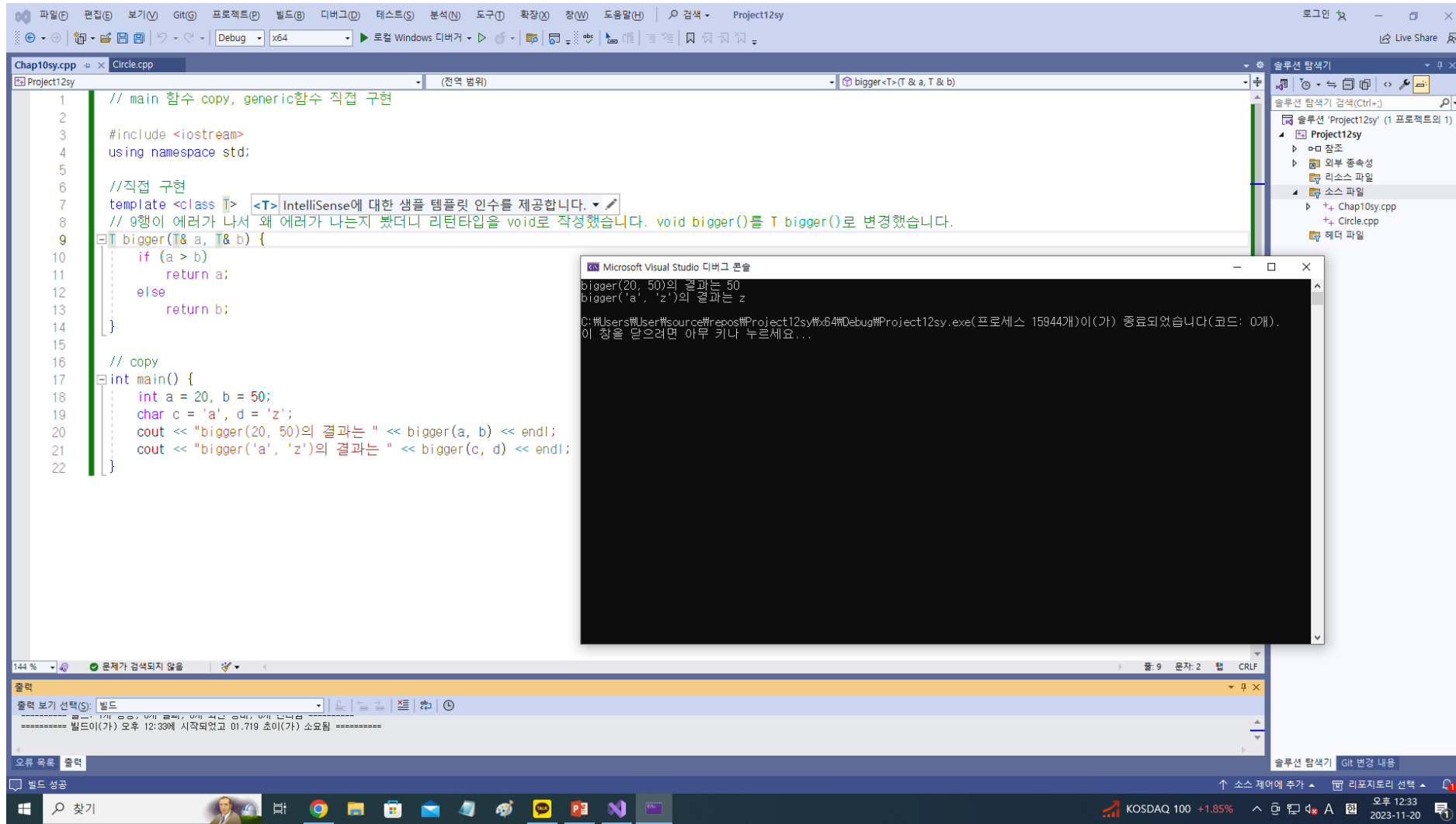
Microsoft Visual Studio 디버깅 콘솔

```
a=5, b=4
c=12.5, d=0.3
donut반지름=20, pizza반지름=5

C:\Users\won2s\Desktop\책체지향프로그래밍\12주차 실습\12주차 실습\64\Debug\12주차 실습.exe (프로세스 8908개)
이(가) 종료되었습니다(코드: 0x#).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔
닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

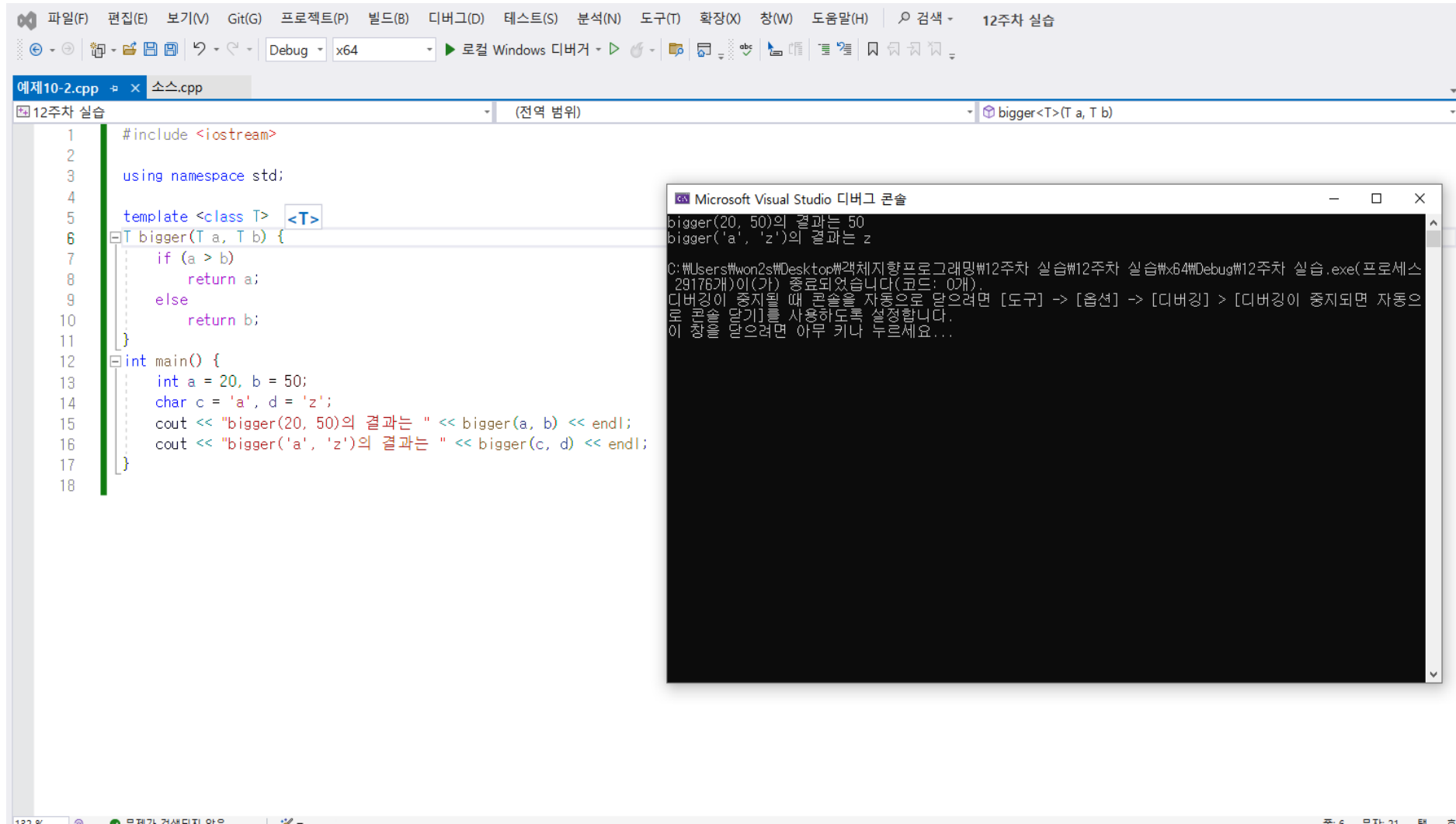
[1차시 이수영] 예제 10-2

main(): copy, template: 안 보고 직접 구현



[1차시 원준서] 예제 10-2

main(): copy, template: 안 보고 직접 구현



The image shows a Visual Studio IDE window with a C++ file named '예제10-2.cpp'. The code defines a template function 'bigger' that compares two values and returns the larger one. The 'main' function tests the function with integers and characters. A debug console window is open, showing the output of the program.

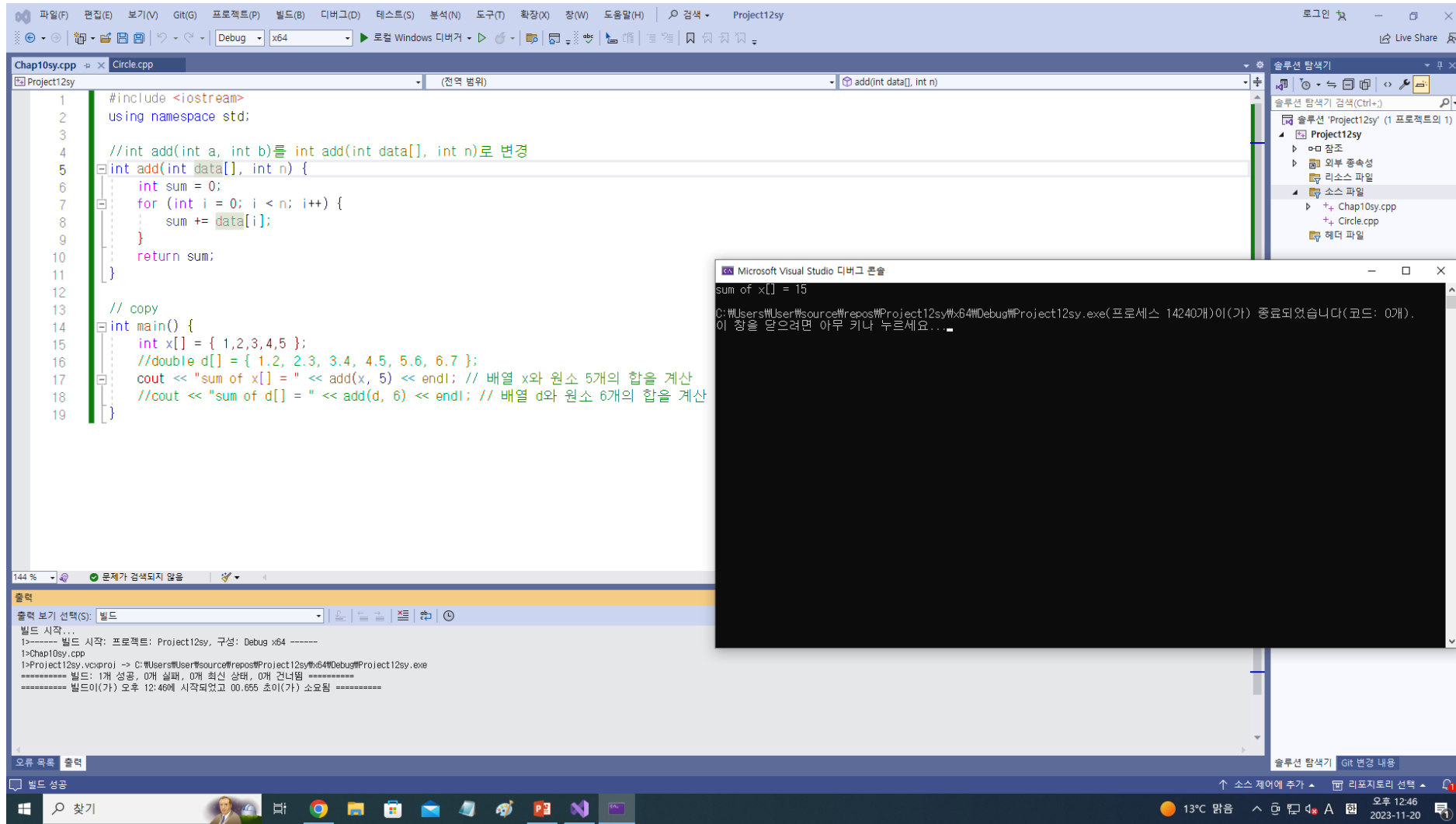
```
#include <iostream>
using namespace std;
template <class T>
T bigger(T a, T b) {
    if (a > b)
        return a;
    else
        return b;
}
int main() {
    int a = 20, b = 50;
    char c = 'a', d = 'z';
    cout << "bigger(20, 50)의 결과는 " << bigger(a, b) << endl;
    cout << "bigger('a', 'z')의 결과는 " << bigger(c, d) << endl;
}
```

Microsoft Visual Studio 디버그 콘솔

```
bigger(20, 50)의 결과는 50
bigger('a', 'z')의 결과는 z
C:\Users\won2s\Desktop\각체지향프로그래밍\12주차 실습\12주차 실습\64\Debug\12주차 실습.exe(프로세스
29176개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으
로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

[1차시 이수영] 예제 10-3

(1) int로 먼저 구현



[1차시 이수영] 예제 10-3

(2) template로 구현

The screenshot displays the Microsoft Visual Studio IDE with a C++ project named 'Project12sy'. The main file, 'Chap10sy.cpp', contains the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 // class 뒤에 T 부분이 늦게 떠오름
5 template <class T> <T> IntelliSense에 대한 샘플 템플릿 인수를 제공합니다.
6
7 //int add(int a, int b)를 int add(int data[], int n)로 변경
8 add(T data[], int n) {
9     T sum = 0;
10    for (int i = 0; i < n; i++) {
11        sum += data[i];
12    }
13    return sum;
14 }
15
16 // copy
17 int main() {
18     int x[] = { 1,2,3,4,5 };
19     double d[] = { 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 };
20     cout << "sum of x[] = " << add(x, 5) << endl; // 배열 x와 원소 5개의 합을 계산
21     cout << "sum of d[] = " << add(d, 6) << endl; // 배열 d와 원소 6개의 합을 계산
22 }
```

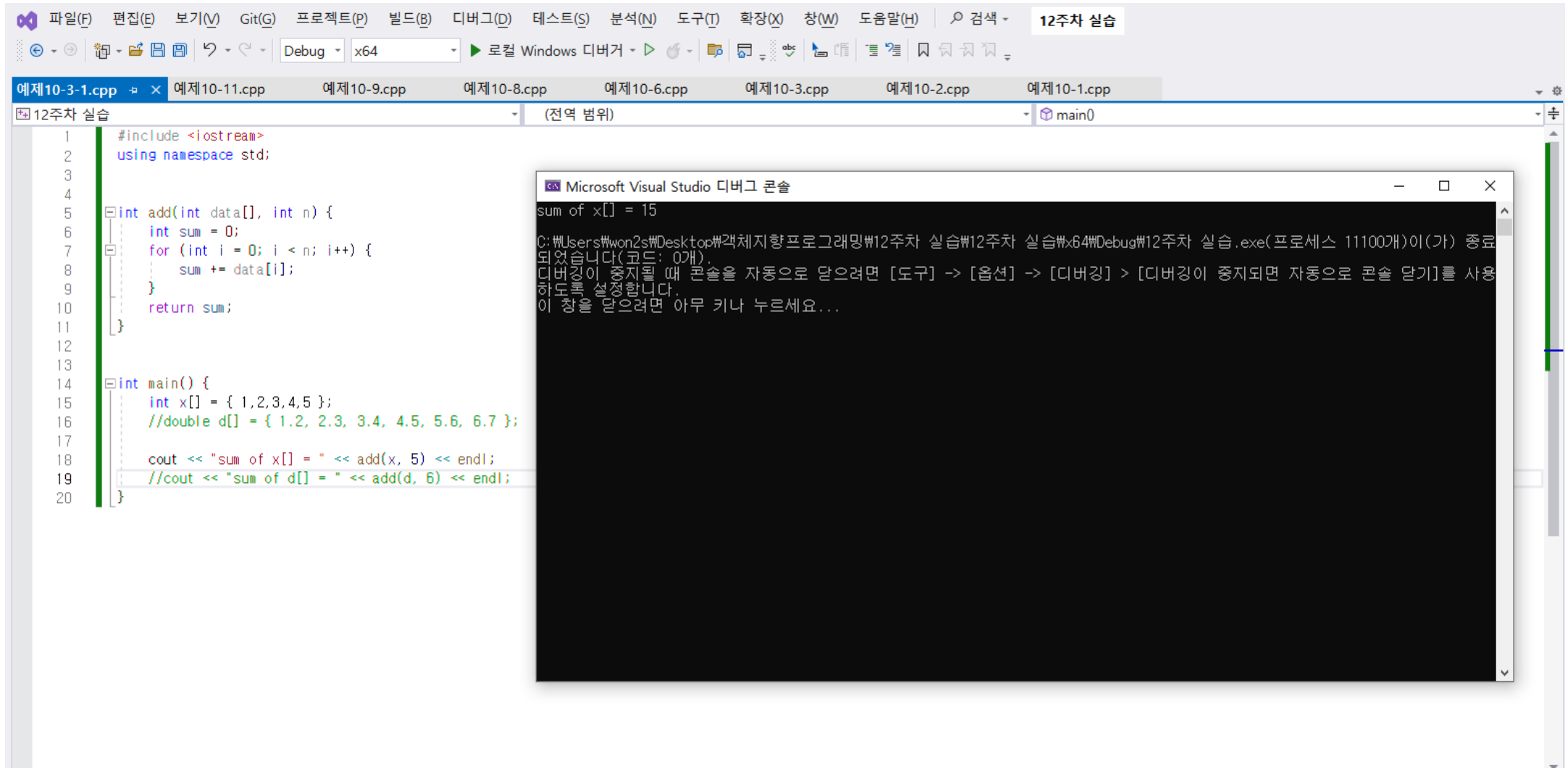
A debug console window is open, showing the output of the program:

```
sum of x[] = 15
sum of d[] = 23.7
C:\Users\User\source\repos\Project12sy\Debug\Project12sy.exe (프로세스 4048개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

The bottom status bar indicates the build was successful: '빌드 성공'.

[1차시 원준서] 예제 10-3

(1) int로 먼저 구현



The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C++ source file named '예제10-3-1.cpp'. The code defines an 'add' function that calculates the sum of an array of integers and a 'main' function that uses this function. A debug console window is open in the foreground, showing the output of the program.

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

int add(int data[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += data[i];
    }
    return sum;
}

int main() {
    int x[] = { 1, 2, 3, 4, 5 };
    //double d[] = { 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 };

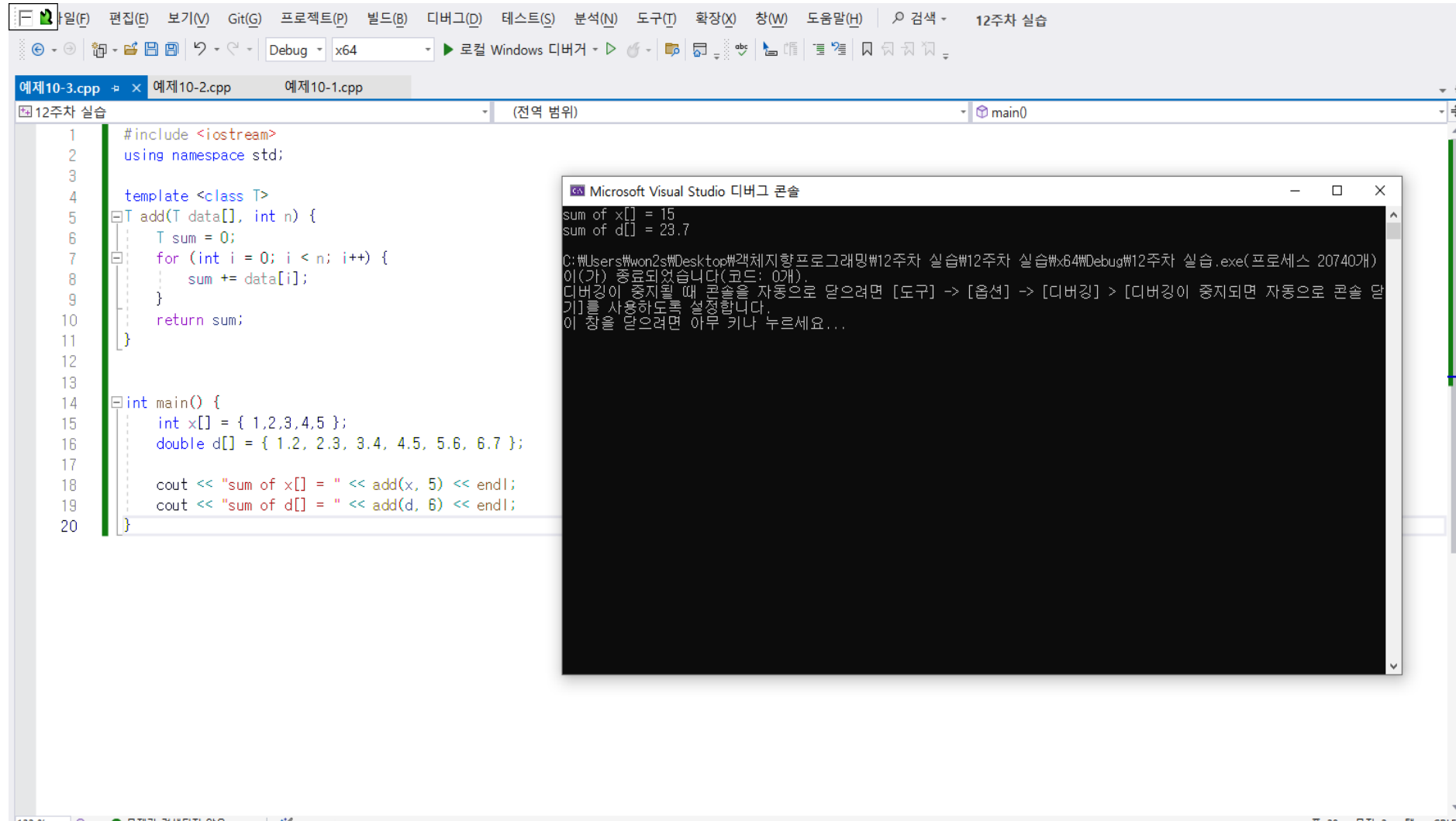
    cout << "sum of x[] = " << add(x, 5) << endl;
    //cout << "sum of d[] = " << add(d, 6) << endl;
}
```

Microsoft Visual Studio 디버그 콘솔

```
sum of x[] = 15
C:\Users\won2s\Desktop\책체지향프로그래밍\12주차 실습\12주차 실습\64\Debug\12주차 실습.exe (프로세스 11100개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

[1차시 원준서] 예제 10-3

(2) template로 구현



The image shows a Visual Studio IDE window with a C++ file named '예제10-3.cpp'. The code defines a template function 'add' that calculates the sum of an array of type 'T'. The 'main' function uses this template to calculate the sum of an integer array 'x' and a double array 'd'. The output is displayed in the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window.

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 T add(T data[], int n) {
6     T sum = 0;
7     for (int i = 0; i < n; i++) {
8         sum += data[i];
9     }
10    return sum;
11 }
12
13
14 int main() {
15     int x[] = { 1,2,3,4,5 };
16     double d[] = { 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 };
17
18     cout << "sum of x[] = " << add(x, 5) << endl;
19     cout << "sum of d[] = " << add(d, 6) << endl;
20 }
```

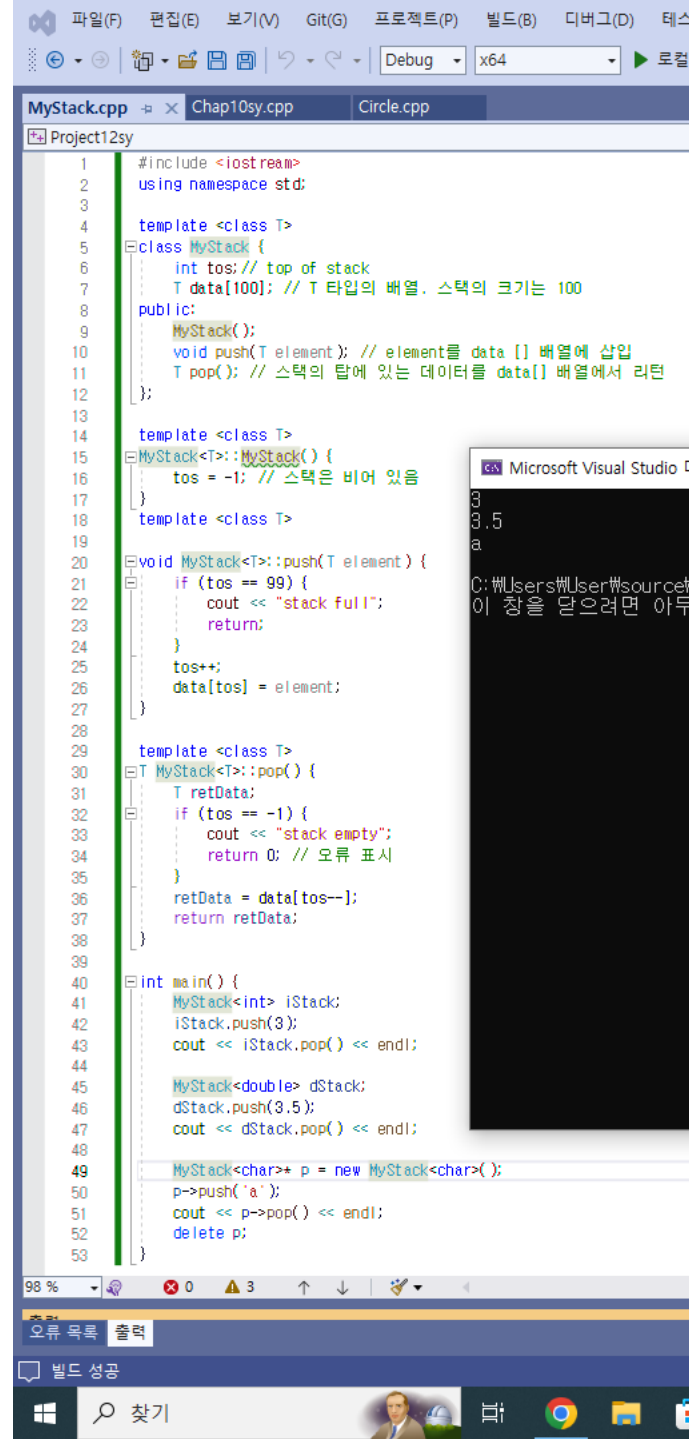
Microsoft Visual Studio 디버그 콘솔

```
sum of x[] = 15
sum of d[] = 23.7

C:\Users\won2s\Desktop\객체지향프로그래밍\12주차 실습\12주차 실습\Debug\12주차 실습.exe (프로세스 20740개)
이(가) 종료되었습니다(코드: 0x0).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

[2차시 이수영] 예제 10-6

제네릭 '클래스' 강조 직접 구현



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

template <class T>
class MyStack {
    int tos; // top of stack
    T data[100]; // T 타입의 배열. 스택의 크기는 100
public:
    MyStack();
    void push(T element); // element를 data [] 배열에 삽입
    T pop(); // 스택의 탑에 있는 데이터를 data[] 배열에서 리턴
};

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

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() {
    if (tos == -1) {
        cout << "stack empty";
        return 0; // 오류 표시
    }
    T retData = data[tos--];
    return retData;
}

int main() {
    MyStack<int> iStack;
    iStack.push(3);
    cout << iStack.pop() << endl;

    MyStack<double> dStack;
    dStack.push(3.5);
    cout << dStack.pop() << endl;

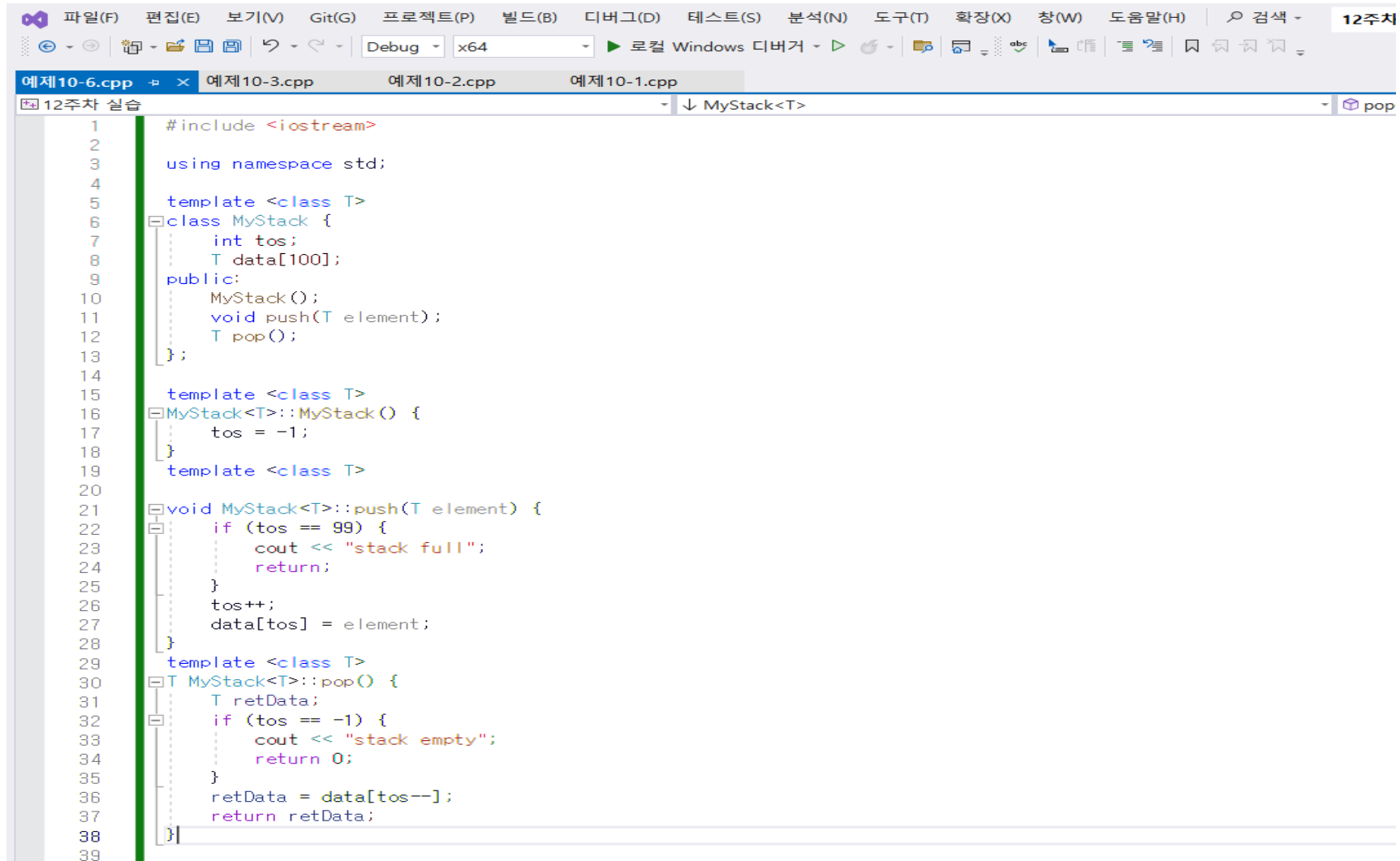
    MyStack<char*> p = new MyStack<char*>();
    p->push('a');
    cout << p->pop() << endl;
    delete p;
}
```

Microsoft Visual Studio C++ Console Output:

```
3
3.5
a
```

[2차시 원준서] 예제 10-6 (1) 코드(1/2)

제네릭 '클래스' 강조 직접 구현



The image shows a screenshot of a C++ code editor with the following code:

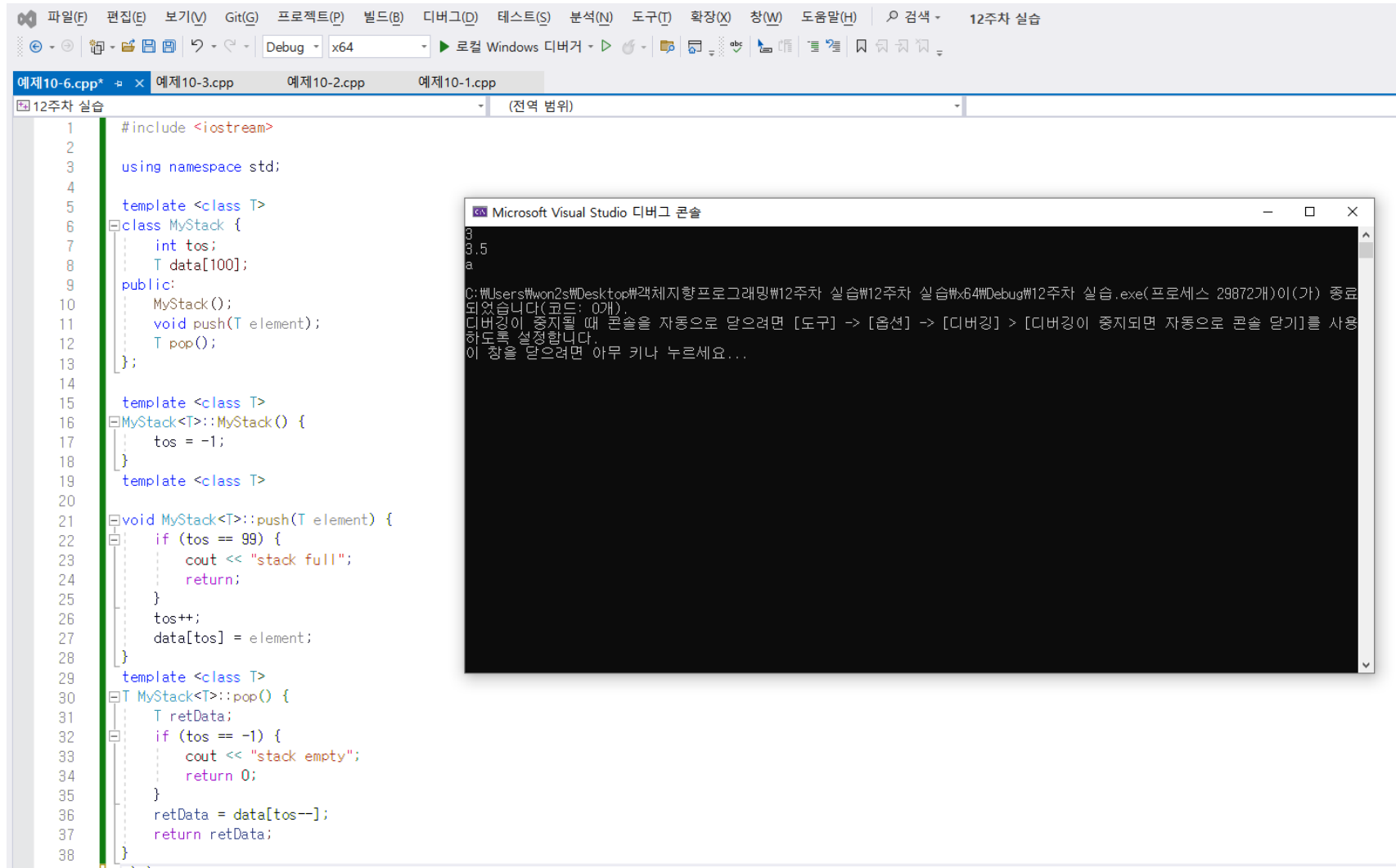
```
1  #include <iostream>
2
3  using namespace std;
4
5  template <class T>
6  class MyStack {
7      int tos;
8      T data[100];
9  public:
10     MyStack();
11     void push(T element);
12     T pop();
13 };
14
15 template <class T>
16 MyStack<T>::MyStack() {
17     tos = -1;
18 }
19
20 template <class T>
21 void MyStack<T>::push(T element) {
22     if (tos == 99) {
23         cout << "stack full";
24         return;
25     }
26     tos++;
27     data[tos] = element;
28 }
29
30 template <class T>
31 T MyStack<T>::pop() {
32     T retData;
33     if (tos == -1) {
34         cout << "stack empty";
35         return 0;
36     }
37     retData = data[tos--];
38     return retData;
39 }
```

[2차시 원준서] 예제 10-6 (2) 코드(2/2)

제네릭 '클래스' 강조 직접 구현

```
39
40 int main() {
41     MyStack<int> iStack;
42     iStack.push(3);
43     cout << iStack.pop() << endl;
44     MyStack<double> dStack;
45     dStack.push(3.5);
46     cout << dStack.pop() << endl;
47     MyStack<char>* p = new MyStack<char>();
48     p->push('a');
49     cout << p->pop() << endl;
50     delete p;
51 }
```

[2차시 원준서] 예제 10-6 (3) 실행결과 제네릭 '클래스' 강조 직접 구현



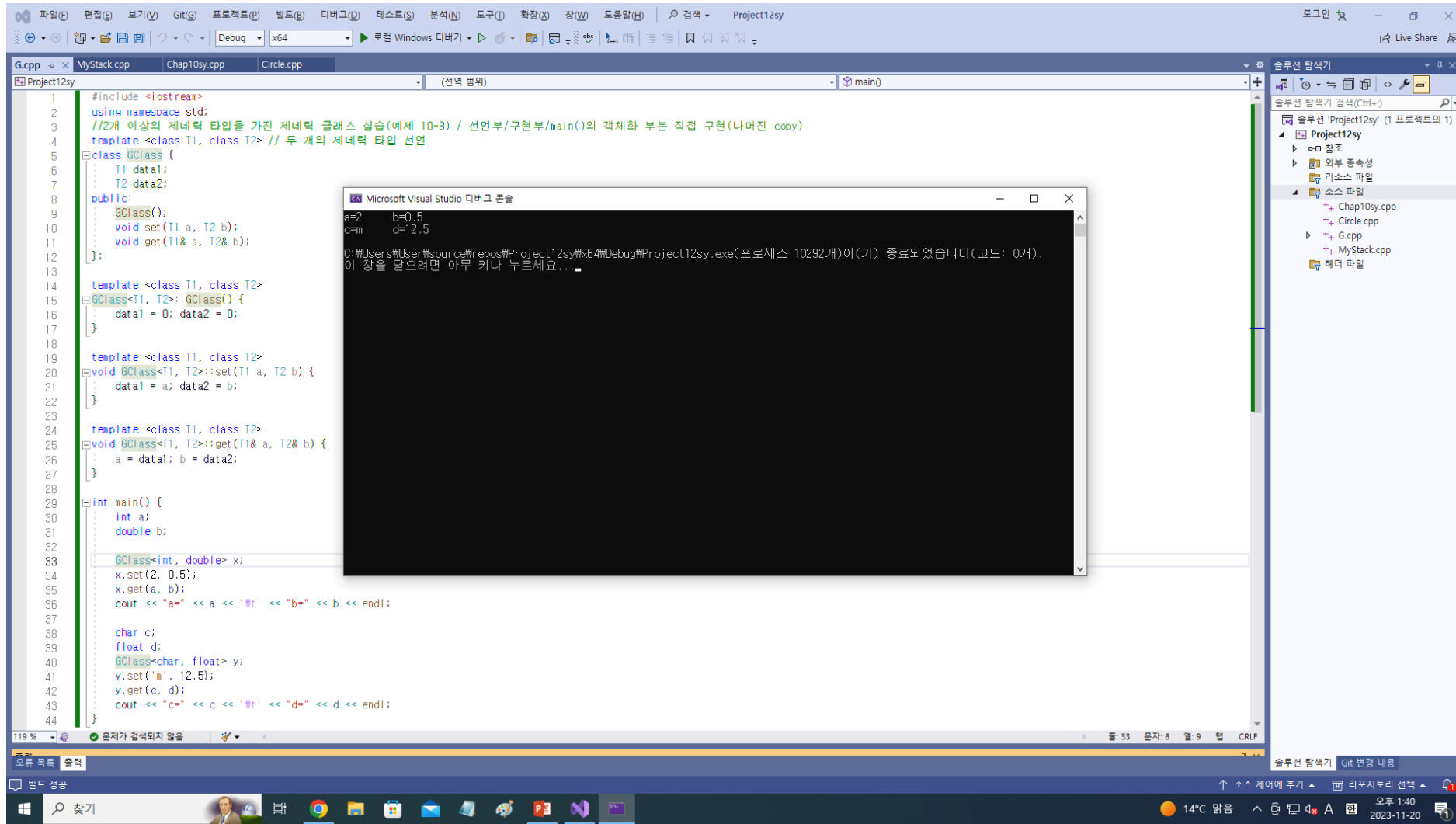
The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C++ source file named '예제10-6.cpp'. The code implements a generic stack class 'MyStack' using templates. The code includes headers, namespace declarations, and methods for push, pop, and initialization. The debug console window is open, showing the output of the program. The output indicates that the stack is full when attempting to push an element, and it also shows the stack's state during execution.

```
1 #include <iostream>
2
3 using namespace std;
4
5 template <class T>
6 class MyStack {
7     int tos;
8     T data[100];
9 public:
10     MyStack();
11     void push(T element);
12     T pop();
13 };
14
15 template <class T>
16 MyStack<T>::MyStack() {
17     tos = -1;
18 }
19
20 template <class T>
21 void MyStack<T>::push(T element) {
22     if (tos == 99) {
23         cout << "stack full";
24         return;
25     }
26     tos++;
27     data[tos] = element;
28 }
29
30 template <class T>
31 T MyStack<T>::pop() {
32     T retData;
33     if (tos == -1) {
34         cout << "stack empty";
35         return 0;
36     }
37     retData = data[tos--];
38     return retData;
39 }
```

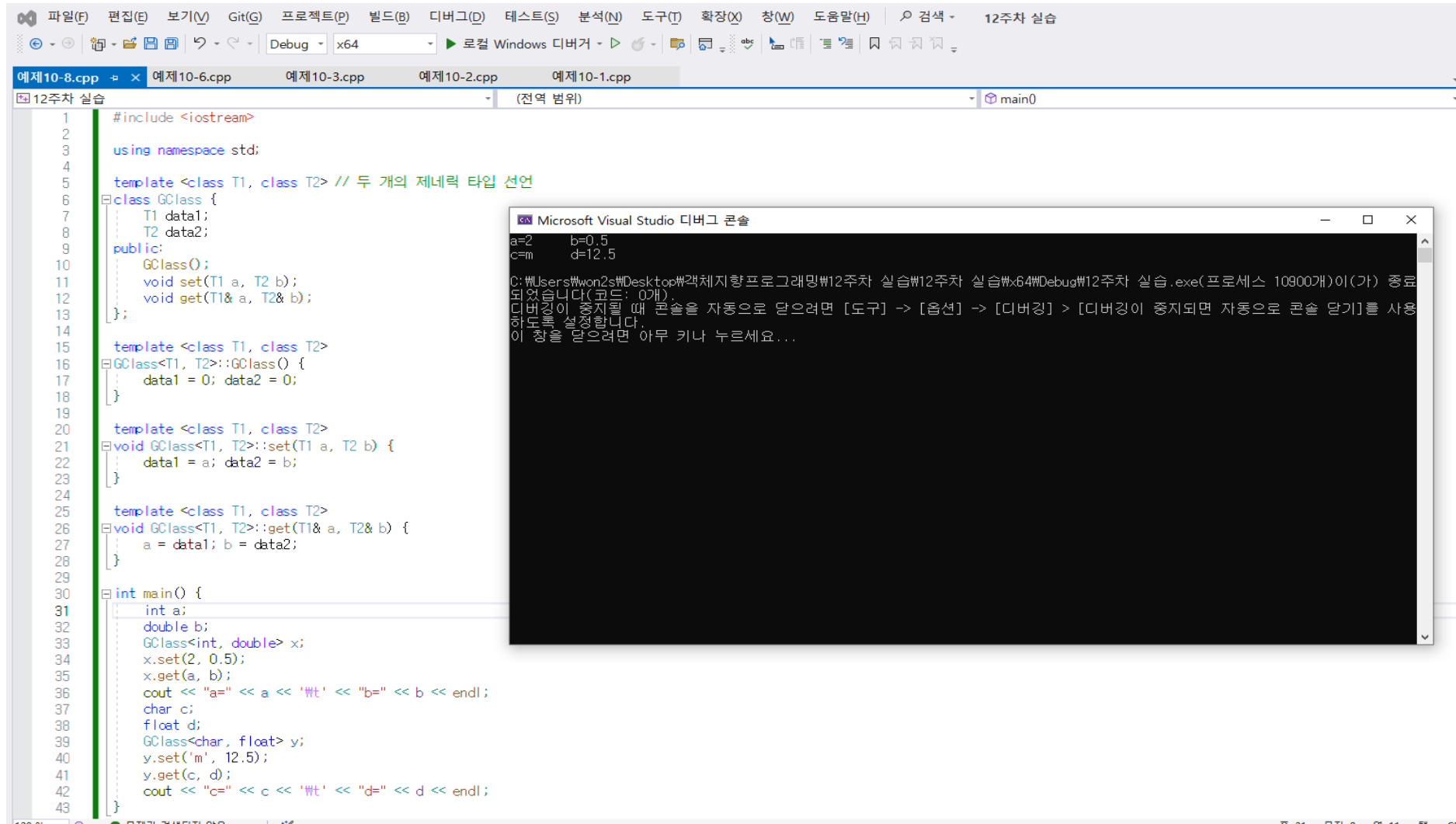
Microsoft Visual Studio 디버깅 콘솔

```
3
3.5
a
C:\Users\won2s\Desktop\객체지향프로그래밍\12주차_실습\12주차_실습\64\Debug\12주차_실습.exe(프로세스 29872개)이(가) 종료
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

[2차시 이수영] 예제8 2개 이상 제네릭 타입 가진 제네릭 클래스 선언/구현/객체화 구현



[2차시 원준서] 예제8 2개 이상 제네릭 타입 가진 제네릭 클래스 선언/구현/객체화 구현



```
1  #include <iostream>
2
3  using namespace std;
4
5  template <class T1, class T2> // 두 개의 제네릭 타입 선언
6  class GClass {
7      T1 data1;
8      T2 data2;
9  public:
10     GClass();
11     void set(T1 a, T2 b);
12     void get(T1& a, T2& b);
13 };
14
15 template <class T1, class T2>
16 GClass<T1, T2>::GClass() {
17     data1 = 0; data2 = 0;
18 }
19
20 template <class T1, class T2>
21 void GClass<T1, T2>::set(T1 a, T2 b) {
22     data1 = a; data2 = b;
23 }
24
25 template <class T1, class T2>
26 void GClass<T1, T2>::get(T1& a, T2& b) {
27     a = data1; b = data2;
28 }
29
30 int main() {
31     int a;
32     double b;
33     GClass<int, double> x;
34     x.set(2, 0.5);
35     x.get(a, b);
36     cout << "a=" << a << '\t' << "b=" << b << endl;
37     char c;
38     float d;
39     GClass<char, float> y;
40     y.set('m', 12.5);
41     y.get(c, d);
42     cout << "c=" << c << '\t' << "d=" << d << endl;
43 }
```

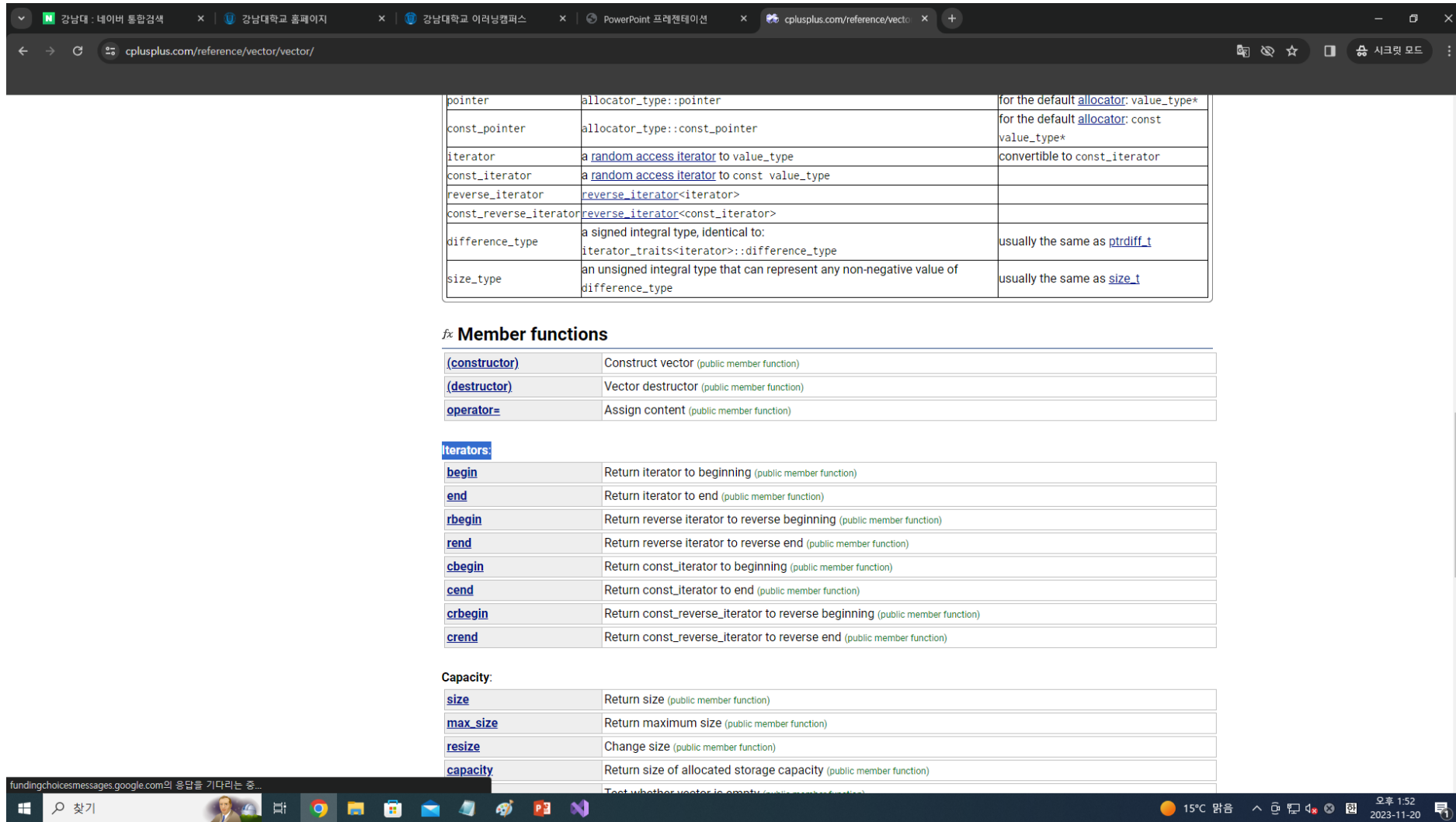
Microsoft Visual Studio 디버깅 콘솔

```
a=2      b=0.5
c=m      d=12.5

C:\Users\won2s\Desktop\객체지향프로그래밍\12주차 실습\12주차 실습\64\Debug\12주차 실습.exe(프로세스 10900개)이(가) 종료
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```


[3차시 이수영] p.21 STL의 예) vector

<https://cplusplus.com/reference/vector/vector/>



The screenshot shows the C++ reference website for the `vector` class. The browser tabs include '강남대 : 네이버 통합검색', '강남대학교 홈페이지', '강남대학교 이러닝캠퍼스', 'PowerPoint 프레젠테이션', and 'cplusplus.com/reference/vector/'. The address bar shows 'cplusplus.com/reference/vector/'.

Member	Description	Notes
<code>pointer</code>	<code>allocator_type::pointer</code>	for the default <code>allocator</code> : <code>value_type*</code>
<code>const_pointer</code>	<code>allocator_type::const_pointer</code>	for the default <code>allocator</code> : <code>const value_type*</code>
<code>iterator</code>	a random access iterator to <code>value_type</code>	convertible to <code>const_iterator</code>
<code>const_iterator</code>	a random access iterator to <code>const value_type</code>	
<code>reverse_iterator</code>	reverse_iterator < <code>iterator</code> >	
<code>const_reverse_iterator</code>	reverse_iterator < <code>const_iterator</code> >	
<code>difference_type</code>	a signed integral type, identical to: <code>iterator_traits<iterator>::difference_type</code>	usually the same as <code>ptrdiff_t</code>
<code>size_type</code>	an unsigned integral type that can represent any non-negative value of <code>difference_type</code>	usually the same as <code>size_t</code>

Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

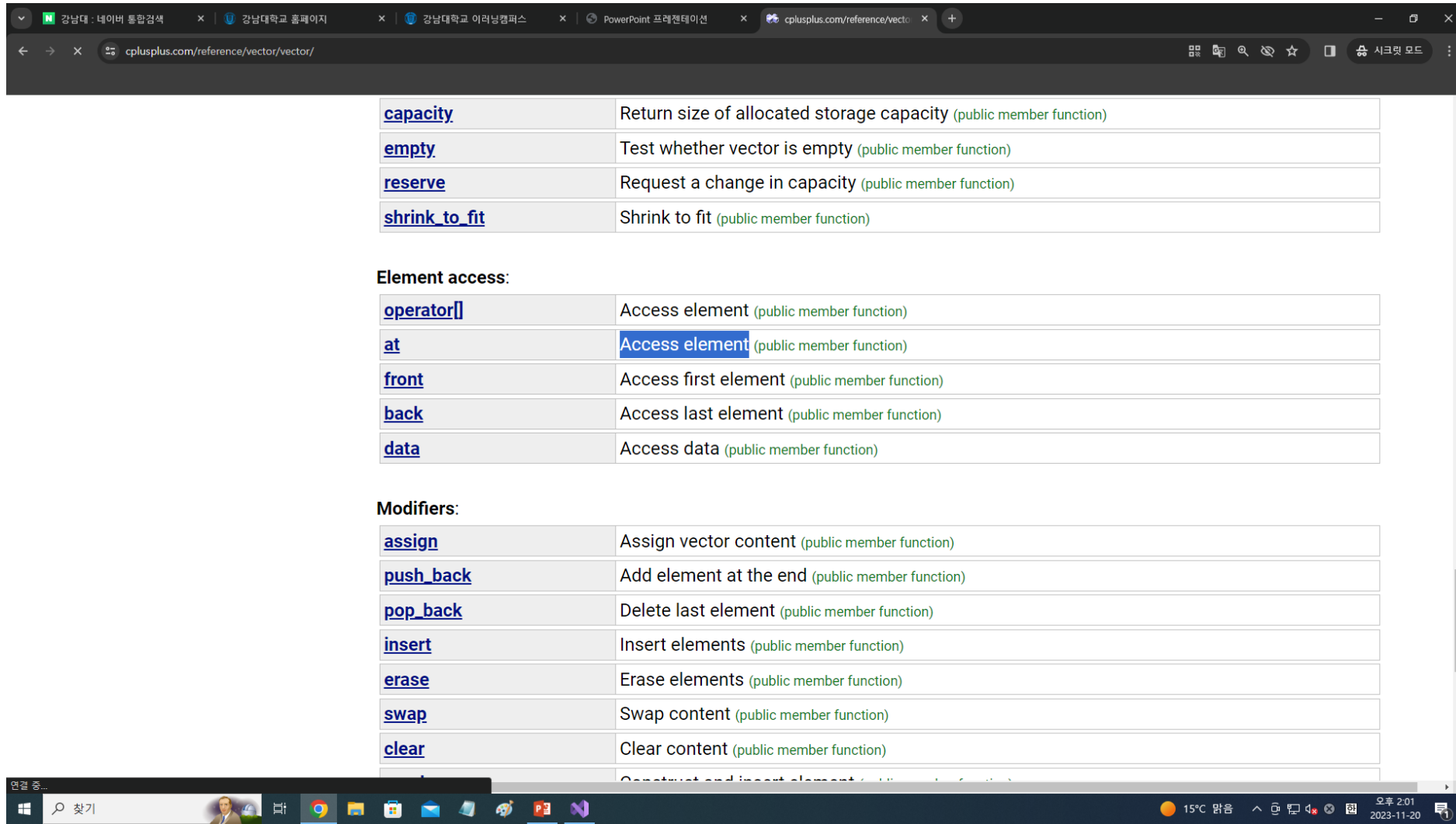
begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin	Return <code>const_iterator</code> to beginning (public member function)
cend	Return <code>const_iterator</code> to end (public member function)
crbegin	Return <code>const_reverse_iterator</code> to reverse beginning (public member function)
crend	Return <code>const_reverse_iterator</code> to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)

[3차시 이수영] p.24 STL의 예) vector

<https://cplusplus.com/reference/vector/vector/>



The screenshot shows a web browser window with the URL <https://cplusplus.com/reference/vector/vector/>. The page lists various member functions of the `vector` class, categorized into capacity, element access, and modifiers.

Function	Description
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit	Shrink to fit (public member function)

Element access:

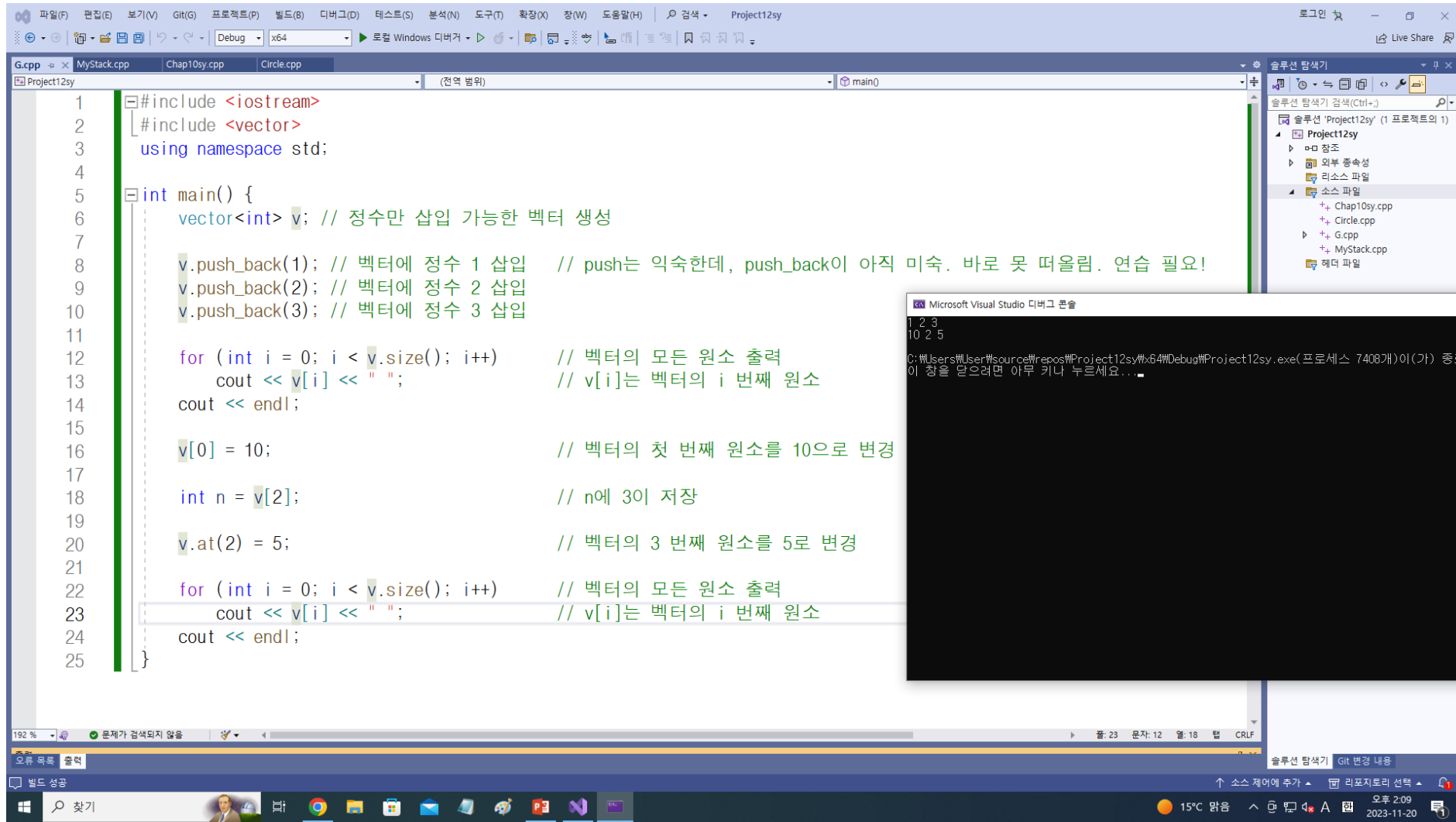
Function	Description
operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data	Access data (public member function)

Modifiers:

Function	Description
assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)

[3차시 이수영] 예제 10-9

STL 이용해서 vector 컨테이너 다뤄보기



The screenshot displays the Visual Studio IDE with a C++ project named 'Project12sy'. The main window shows the source code for 'G.cpp', which demonstrates the use of the `std::vector` container. The code includes `<iostream>` and `<vector>`, and uses the `std` namespace. The `main` function performs the following steps:

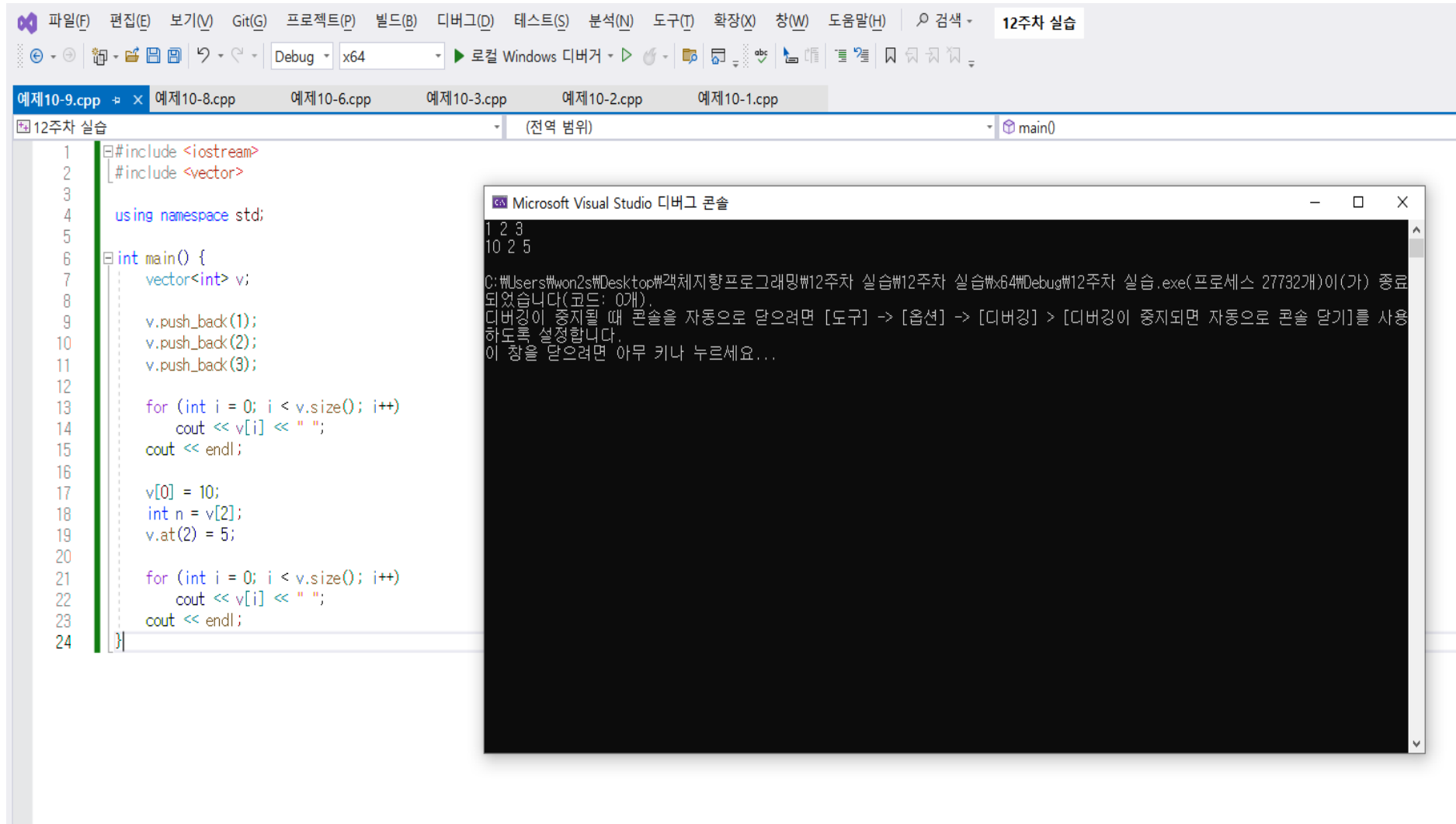
- Creates a `vector<int>` named `v`.
- Pushes back the integers 1, 2, and 3 into the vector.
- Iterates over the vector and prints each element, resulting in the output: `1 2 3`.
- Changes the first element (`v[0]`) to 10.
- Stores the value at index 2 (`v[2]`) in a variable `n`, which is 3.
- Changes the element at index 2 (`v[2]`) to 5.
- Iterates over the vector again and prints each element, resulting in the output: `10 2 5`.

A debug console window is open in the bottom right, showing the execution output: `1 2 3` and `10 2 5`. The status bar at the bottom indicates the file is 23 lines long, 12 columns wide, and uses the CRLF line ending.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<int> v; // 정수만 삽입 가능한 벡터 생성
7
8     v.push_back(1); // 벡터에 정수 1 삽입 // push는 익숙한데, push_back이 아직 미숙. 바로 못 떠올림. 연습 필요!
9     v.push_back(2); // 벡터에 정수 2 삽입
10    v.push_back(3); // 벡터에 정수 3 삽입
11
12    for (int i = 0; i < v.size(); i++) // 벡터의 모든 원소 출력
13        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
14    cout << endl;
15
16    v[0] = 10; // 벡터의 첫 번째 원소를 10으로 변경
17
18    int n = v[2]; // n에 3이 저장
19
20    v[2] = 5; // 벡터의 3 번째 원소를 5로 변경
21
22    for (int i = 0; i < v.size(); i++) // 벡터의 모든 원소 출력
23        cout << v[i] << " "; // v[i]는 벡터의 i 번째 원소
24    cout << endl;
25 }
```

[3차시 원준서] 예제 10-9

STL 이용해서 vector 컨테이너 다뤄보기



The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C++ source file named '예제10-9.cpp'. The code uses the STL `vector` container. It includes `<iostream>` and `<vector>`, uses the `std` namespace, and defines a `main` function. In `main`, a `vector<int>` named `v` is created. Three integers (1, 2, 3) are pushed back into the vector. Then, the vector's contents are printed using a `for` loop. After this, the first element is set to 10, the third element is set to 5, and the vector's contents are printed again.

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> v;
8
9     v.push_back(1);
10    v.push_back(2);
11    v.push_back(3);
12
13    for (int i = 0; i < v.size(); i++)
14        cout << v[i] << " ";
15    cout << endl;
16
17    v[0] = 10;
18    int n = v[2];
19    v.at(2) = 5;
20
21    for (int i = 0; i < v.size(); i++)
22        cout << v[i] << " ";
23    cout << endl;
24 }
```

The 'Debug Console' window is open, showing the output of the program. It displays the first print statement's output '1 2 3' on one line and the second print statement's output '10 2 5' on the next line. Below the output, there is a message in Korean indicating that the program has finished execution successfully.

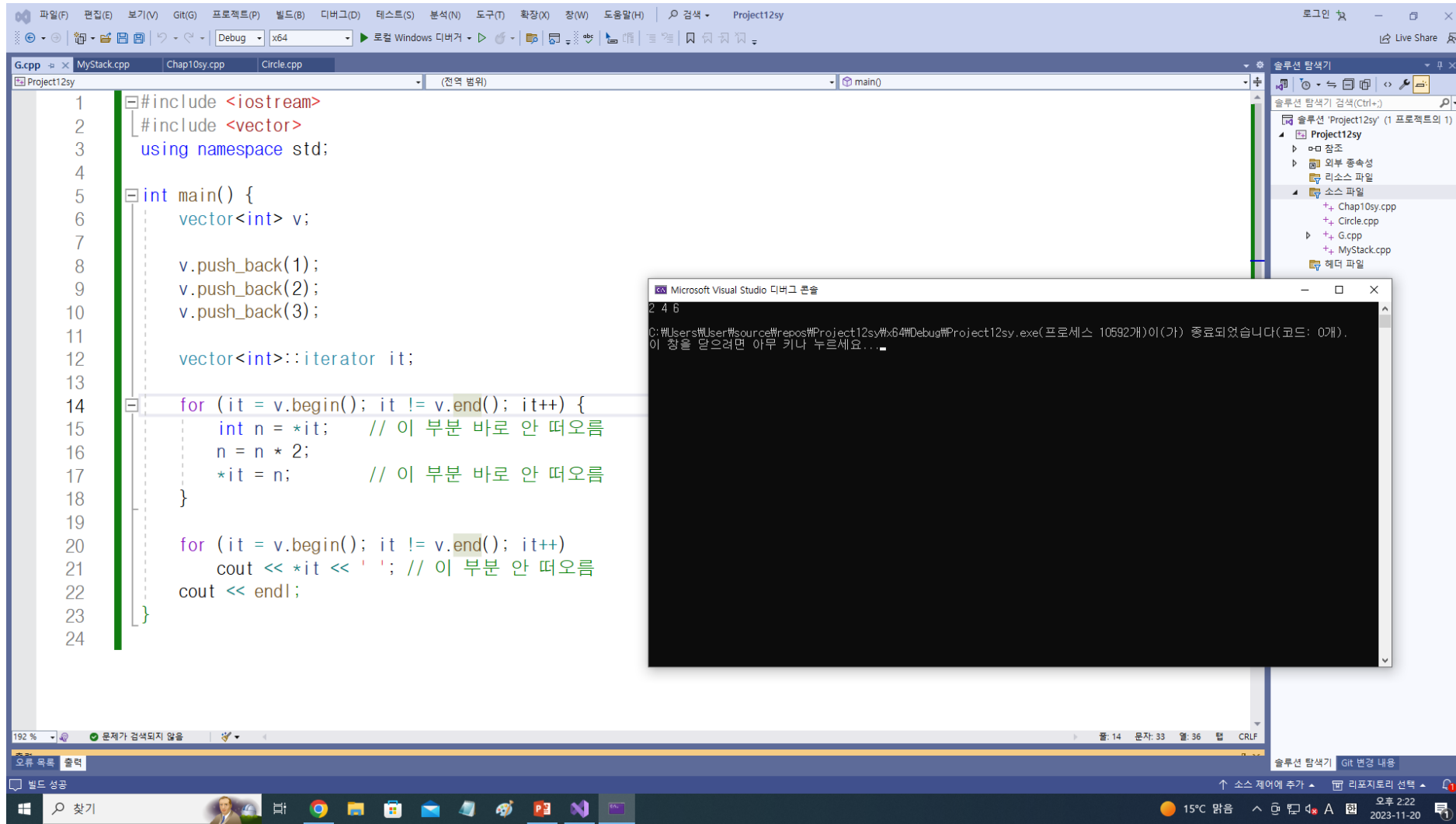
Microsoft Visual Studio 디버그 콘솔

```
1 2 3
10 2 5

C:\Users\won2s\Desktop\객체지향프로그래밍\12주차 실습\12주차 실습\Debug\12주차 실습.exe(프로세스 27732개)이(가) 종료
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

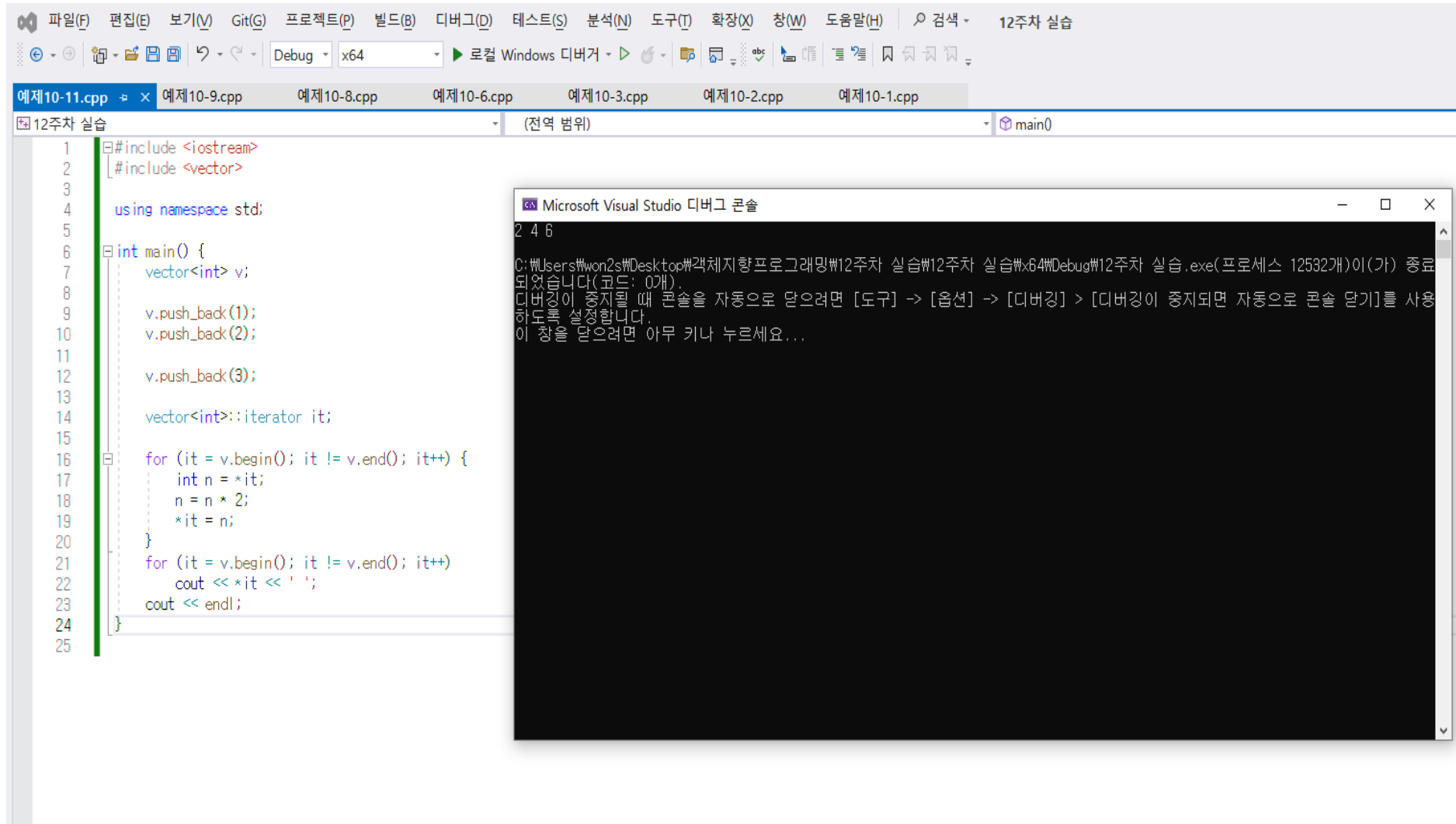
[3차시 이수영] 예제 10-11

iterator 사용 vector



[3차시 원준서] 예제 10-11

iterator 사용 vector



The image shows a Visual Studio IDE with a C++ file named '예제10-11.cpp' open. The code defines a vector 'v' and iterates over its elements using a range-based for loop, doubling each value and printing it. A debug console window is open, showing the output '2 4 6' and a message indicating the program has finished execution.

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int main() {
7     vector<int> v;
8
9     v.push_back(1);
10    v.push_back(2);
11
12    v.push_back(3);
13
14    vector<int>::iterator it;
15
16    for (it = v.begin(); it != v.end(); it++) {
17        int n = *it;
18        n = n * 2;
19        *it = n;
20    }
21    for (it = v.begin(); it != v.end(); it++)
22        cout << *it << ' ';
23    cout << endl;
24 }
25
```

Microsoft Visual Studio 디버그 콘솔

2 4 6

C:\Users\won2s\Desktop\그래프체지향프로그래밍\12주차 실습\12주차 실습\Debug\12주차 실습.exe(프로세스 12532개)이(가) 종료되었습니다(코드: 0개).

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...

[정리노트 원준서]

일반화와 템플릿

- 제네릭 또는 일반화는 함수나 클래스를 일반화시키고, 매개 변수 타입을 지정하여 틀에서 찍어 내듯이 함수나 클래스 코드를 생산하는 기법입니다.
- 템플릿은 함수나 클래스를 일반화하는 C++ 도구로, `template` 키워드로 함수나 클래스를 선언합니다. 변수나 매개 변수의 타입만 다르고, 코드 부분이 동일한 함수를 일반화시킵니다. 제네릭 타입은 일반화를 위한 데이터 타입입니다.
- 템플릿 선언은 다음과 같이 선언합니다.
`template <class T>` 또는 `template <typename T>`
3개의 제네릭 타입을 가진 템플릿 선언은 다음과 같습니다.
`template <class T1, class T2, class T3>`

[정리노트 이수영]

일반화와 템플릿

- 매개변수만 다르고 함수이름은 같은 오버로딩은 동일한 코드가 중복 작성되는 한계가 있는데, 일반화 과정이 필요하다.
- 일반화(generic)는 `template` 키워드로 함수/클래스를 선언하는 템플릿을 통해 구현할 수 있다.
- `template` 키워드를 쓰고 꺾쇠(<) 제네릭 타입을 선언하는 `class` 제네릭 타입 `T` 선언 후에 꺾쇠(>)로 닫는다.
- `template <class T>`는 예제 10-6에서 확인할 수 있다시피 매번 작성 해주어야 한다. `T`는 `int`, `double`, `float` 등의 여러 타입을 일반화한다.
- 여러 매개타입을 쓰는 즉 `T1`, `T2`, `T3`으로 표현이 가능한데 이 때는 콤마(,)를 사용해서 `template <class T1, T2, T3>`과 같이 표현하면 된다.

[정리노트 원준서]

템플릿으로부터의 구체화

- 구체화는 템플릿의 제네릭 타입에 구체적인 타입을 지정하며, 템플릿 함수로부터 구체화된 함수의 소스 코드를 생성합니다.
- 제네릭 타입에 구체적인 타입 지정 시 주의해야 하는데, 주의하지 않을 시 구체화 오류가 발생할 수도 있습니다.

[정리노트 이수영]

템플릿으로부터의 구체화

- 템플릿 선언부는 T로 작성하지만, main()과 같은 구현부에서는 T가 아니라 직접적인/구체적인 데이터 타입을 명시해야 한다.
- void myswap(T &a, T &b)로 템플릿 함수를 선언했다면, 이를 구현하는 코드에서는 void myswap(int &a, int &b)와 같이 구체적으로 명시해야 컴파일러가 오류 없이 제대로 컴파일할 수 있게 된다.
- 그런데 T1, T2가 아니라 T로 제네릭 타입을 선언했다면, 들어올 변수 데이터타입은 int, int 또는 double, double이어야지, 여러 타입이 섞인 int, double의 경우 오류가 날 수 있으니 유의한다.

[정리노트 원준서]

템플릿 장점&제네릭 프로그래밍

- 템플릿의 장점은 함수 코드의 재사용으로, 높은 소프트웨어의 생산성과 유용성입니다. 템플릿의 단점으로는 포팅에 취약하며, 컴파일러에 따라 지원하지 않을 수 있습니다. 컴파일 오류 메시지가 빈약하며, 디버깅에 많은 어려움이 있습니다.
- 제네릭 프로그래밍은 일반화 프로그래밍이라고도 부르며, 제네릭 함수나 제네릭 클래스를 활용하는 프로그래밍 기법입니다. C++에서 STL을 제공하고, 활용합니다. Java, C#등 많은 언어에서 활용됩니다.

[정리노트 이수영]

템플릿 장점&제네릭 프로그래밍

- 템플릿 개념을 모르는 경우에 가독성은 떨어질 수 있지만, 같은 코드를 중복하지 않아도 된다는 장점이 있다.
- 3차시 때 배운 부분 중에 가장 기억에 남는 부분 중 하나가 STL 인데, STL을 충분히 숙지한다면 제네릭 프로그래밍 덕분에 코드 짜는 과정이 시간 지연 없이 효율적으로 진행될 것 같다.
- 제네릭 함수/클래스를 쓰는 제네릭 프로그래밍은 C++뿐 아닌 Java에서도 활용되고 있다. API를 자주 확인하면서, 쓸 수 있는 코드는 잘 사용하는 것도 좋은 프로그래밍 기법임을 배웠다.

[정리노트 원준서]

C++ 표준 템플릿 라이브러리 STL

- STL은 표준 템플릿 라이브러리로, C++ 표준 라이브러리 중 하나입니다. 많은 제네릭 클래스와 제네릭 함수를 포함하기에 개발자는 이들을 활용하여 쉽게 응용 프로그램을 작성할 수 있습니다.
- STL의 구성은 다음과 같습니다.
- 컨테이너 - 템플릿 클래스

데이터를 담아두는 자료 구조를 표현한 클래스로, 리스트, 큐, 스택, 맵, 셋, 벡터가 해당됩니다.

- iterator - 컨테이너 원소에 대한 포인터

컨테이너의 원소들을 순회하면서 접근하기 위해 만들어진 컨테이너 원소에 대한 포인터입니다

- 알고리즘 - 템플릿 함수

컨테이너 원소에 대한 복사, 검색, 삭제, 정렬 등의 기능을 구현한 템플릿 함수입니다.
컨테이너의 멤버 함수가 아닙니다.

[정리노트 이수영]

C++ 표준 템플릿 라이브러리 STL

- STL은 컨테이너, iterator, 알고리즘으로 분류된다.
- 컨테이너는 자료구조(linked list, queue, stack, **vector**, **map** 등)를 표현하는 클래스다.
- iterator는 중요한 개념인데, 컨테이너의 컴포넌트(원소)에 대한 포인터다. 그래서 컨테이너의 컴포넌트에 접근할 때 쓰인다.
표가 기억 나는데 iterator는 전진, **reverse_iterator**는 후진이다.
- 알고리즘은 copy, search, find, sorting 등을 구현하는 함수다.

[정리노트 원준서]

STL과 관련된 헤더파일&이름공간

- 컨테이너 클래스를 활용하기 위한 헤더 파일입니다.
- 해당 클래스가 선언된 헤더 파일은 include입니다.
- ex) vector클래스를 사용하려면 `#include <vector>`, list클래스를 사용하려면 `#include <list>`
- 알고리즘 함수를 사용하기 위한 헤더 파일은 알고리즘 함수에 상관 없이 `#include <algorithm>`입니다.
- STL이 선언된 이름 공간은 std입니다.

[정리노트 이수영]

STL과 관련된 헤더파일&이름공간

- STL의 구성 요소는 컨테이너, iterator, 알고리즘인데 이를 사용하기 위해서는 헤더파일&이름공간을 잘 선언해주어야 한다.
- 컨테이너를 사용하려면 `#include <vector>`와 같이 쓰면 된다.
- 알고리즘을 사용하려면 `#include <algorithm>`을 쓰면 된다.
알고리즘은 각각 자료구조 클래스명을 적어야 하는 컨테이너와 달리, `<algorithm>`만 써주면 모든 알고리즘 함수를 쓸 수 있다.
- 헤더파일은 이렇게 선언하면 되고, 이름공간은 `std`만 쓰면 된다.
`using namespace std;` 이렇게 선언해주면 다 사용할 수 있다.

[정리노트 원준서]

(1) vector 컨테이너

- vector 컨테이너의 특징으로는 가변 길이 배열을 구현한 제네릭 클래스이기에, 개발자가 벡터의 길이에 대한 고민이 필요 없습니다. 원소의 저장, 삭제, 검색 등 다양한 멤버 함수를 지원하며, 벡터에 저장된 원소는 인덱스로 접근 가능합니다.
- ※인덱스는 0부터 시작합니다.

[정리노트 이수영]

(1) vector 컨테이너

- vector는 배열은 배열인데 가변 길이의 배열을 구현한 클래스다. 특히 `push_back()`이 실습 과정에서 놓쳤던 부분이라 가장 기억난다. `push()`, `pop()`이 익숙해서 `_back()`을 까먹고 못 쓰게 된다. 가장 기억에 남는 함수는 `at(int index)`, `begin()`, `end()`다. `begin()`, `end()`는 항상 같이 붙어 다녀서 기억나고, `at`은 `index`에 접근하는 방법이라 기억난다. `push_back()`을 잘 기억하는 방법을 떠올려봤는데, `push`는 원소(`element`)를 추가한다는 의미지만, 벡터의 마지막 즉 뒷부분에 추가하는 거니까 `_back`까지 붙여준다 생각하면 안 까먹을 것 같다. `Begin()`은 벡터의 첫번째 원소를 참조하고, `end()`는 벡터의 마지막 원소 자체가 아닌 그 다음을 가리킨다는 점에 유의해야 할 것 같다.

[정리노트 원준서]

iterator 사용

- iterator는 반복자라고도 부르며, 컨테이너의 원소를 가리키는 포인터입니다.
- iterator 변수 선언은 구체적인 컨테이너를 지정하여 반복자 변수를 생성합니다.

[정리노트 이수영]

iterator 사용

- STL의 구성 요소인 컨테이너, iterator, 알고리즘 중 가장 중요한 iterator는 '포인터' 개념이다. 그래서 '주소'를 가리키게 된다.
- `vector<int> v;`에서 `vector<int>::iterator it; it=v.begin();`일 경우 it는 첫번째 주소를 가리킨다. 그리고 `v.end()`를 유의해야 한다.
- 수업을 정확히 이해하기 전에 `v.end()`를 막연히 떠올려봤을 때 마지막 인덱스의 주소를 가리킨다고 생각했는데, 마지막 인덱스 '그 다음' 인덱스의 주소를 가리킨다는 점을 깨달았다. 한 번 놓쳤던 부분이라 오래 기억날 것 같다.
- 특히 '원소 읽기' 방법이 기억에 남는데, `it++;`을 사용한 부분이다. it가 포인터이기 때문에 ++을 사용해서 그 다음주소로의 접근이 가능하다는 점이 새롭고 효율적인 방법이라고 느꼈다.
- for문을 사용해서 `v.begin()`부터 `v.end()`가 아닐 때까지 it를 ++하는 걸 기억하자!

[정리노트 원준서]

(2) map 컨테이너

- ('키', '값')의 쌍을 원소로 저장하는 제네릭 컨테이너입니다. 동일한 '키'를 가진 원소가 중복 저장되면 오류가 발생합니다.
- '키'로 '값'을 검색하며, 많은 응용에서 필요합니다. `#include <map>`을 필요로 합니다.

[정리노트 이수영]

(2) map 컨테이너

- vector 컨테이너 다음으로 다룬 map 컨테이너
- vector 컨테이너가 인덱스로 데이터에 접근하고, map 컨테이너는 (키, 값) 쌍이 하나의 원소라, 키로 데이터(값)를 접근/검색한다.
- `#include <vector>`를 썼던 것처럼, 알고리즘이 아닌 컨테이너기 때문에 `#include <map>`을 선언해주면 된다.
- 원소를 검색하는 방법은 어떤 원소가 저장됐냐에 따라 다른데, `dic.insert(makePair("love", "사랑"));`으로 저장되면, `dic["love"];` `dic["love"] = "사랑";`으로 저장되면, `dic.at("love");`으로 검색된다. `.at`을 사용하면 index 대신 key로 검색할 수 있다. vector의 `at`은 index로 접근하는 반면, map의 `at`은 key로 접근하는 게 다르다.

[정리노트 원준서]

STL 알고리즘 사용하기

- 알고리즘 함수는 템플릿 함수, 전역 함수이며 STL 컨테이너 클래스의 멤버 함수가 아닙니다. iterator와 함께 작동합니다.

[정리노트 이수영]

STL 알고리즘 사용하기

- STL의 T에서 알 수 있듯, STL은 템플릿 함수이며, 전역 함수다. 전역 함수의 장점은 객체를 따로 생성하지 않아도 전역함수를 불러오면 바로 사용할 수 있다는 점이다.
- 예를 들어 `vector<int> v;`일 때 만일 벡터 전체를 정렬하려면 `sort(v.begin(), v.end());`를 사용하면 된다. 전역함수라는 점에서 `sort`를 사용하기 위한 사전 작업을 하지 않고도 바로 `sort()`를 쓸 수 있다는 장점이 있다. local 변수와의 차이점이다.
- 단, 알고리즘이므로 `#include <algorithm>`을 꼭 선언해야 한다.

[정리노트 원준서]

C++에서의 auto

- C++에서 auto의 기능은 다음과 같습니다.
- C++ 11부터 auto 선언의 의미 수정 : 컴파일러에게 변수 선언 문에서 추론하여 타입을 자동 선언하도록 지시합니다.
- C++ 11 이전까지는 스택에 할당되는 지역 변수를 선언하는 키워드입니다.
- 장점으로는 복잡한 변수 선언을 간소하게, 긴 타입 선언 시 오타를 줄여줍니다.

[정리노트 이수영]

C++에서의 auto

- auto 키워드를 사용하면 데이터 타입을 일일이 지정 안 해도 알아서 등호 뒤의 값을 보고 자동으로 데이터타입이 추론된다.
- 예를 들어 `auto pi = 3.14;`는 `double` 타입으로,
`auto n = 3;`은 `int` 타입으로,
`auto *p = &n;`은 변수 `p`가 `int*` 타입으로 추론된다.
앞서 `n`이 3으로 정수였기 때문이다.
- 이처럼 데이터타입이 서로 다른데 그걸 직접 손으로 작성하는 과정에서 나올 수 있는 실수 또는 번거로움을 해결해준다.