

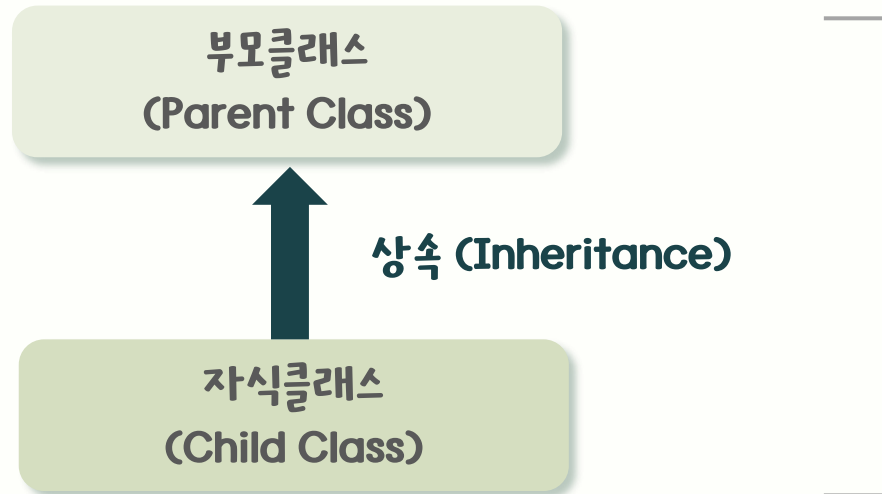
Seminar Presentation

휴켄 S/W개발부 연구원 김하윤

상속 (Inheritance)

상속

객체 지향 프로그래밍의 주요한 특성 중 하나이며 이를 통해 프로그램의 논리적 구조를 계층적으로 구성



소스코드의 재사용성을 높임.

상속 (Inheritance)의 사용

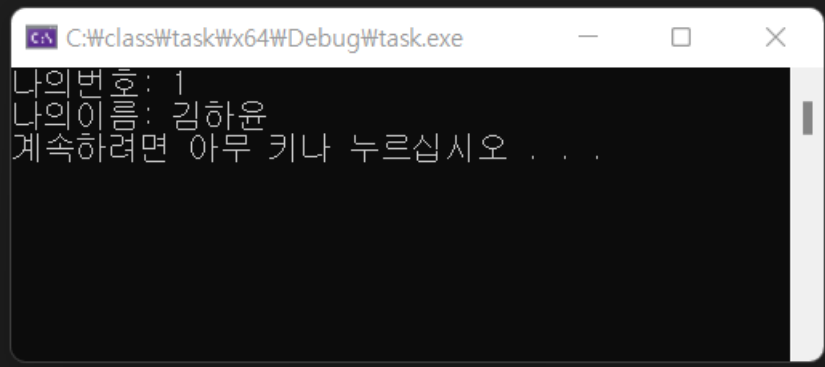
```
#include <iostream>
#include <string>

using namespace std;

class Person {
private:
    string name;
public:
    Person(string name) : name(name) {}
    string getName() {
        return name;
    }
    void showName() {
        cout << "나의이름: " << getName() << '\n';
    }
};
```

```
class Myname : Person {
private:
    int Mynumber;
public:
    Myname(int Mynumber, string name) : Person(name) {
        this->Mynumber = Mynumber;
    }
    void show() {
        cout << "나의번호: " << Mynumber << '\n';
        cout << "나의이름: " << getName() << '\n';
    }
};

int main(void) {
    Myname myname(1, "김하윤");
    myname.show();
}
```



A screenshot of a Windows command prompt window titled "C:\Wclass\Task\Wx64\Debug\Task.exe". The window displays the output of the program: "나의번호: 1", "나의이름: 김하윤", and "계속하려면 아무 키나 누르십시오 . . .".

오버라이딩 (Overriding)

오버라이딩

부모클래스에서 정의된 함수를 자식 클래스에서 재정의하는 문법.

오버라이딩을 적용한 함수의 원형은 기존의 함수와 동일한 매개변수를 전달 받는다.

```
#include <iostream>
#include <string>

using namespace std;

class Person {
private:
    string name;
public:
    Person(string name) : name(name) { }
    string getName() {
        return name;
    }
    void showName() {
        cout << "나의 이름: " << getName() << '\n';
    }
};
```

```
C:\class\task\wx64\Debug\task.exe
나의이름: 김하윤
계속하려면 아무 키나 누르십시오 . . .
```

```
class Myname : Person {
private:
    int Mynumber;
public:
    Myname(int Mynumber, string name) : Person(name) {
        this->Mynumber = Mynumber;
    }
    void show() {
        cout << "나의번호: " << Mynumber << '\n';
        cout << "나의이름: " << getName() << '\n';
    }
    //오버라이드
    void showName() {
        cout << "나의이름: " << getName() << '\n';
    }
};

int main(void) {
    Myname myname(1, "김하윤");
    myname.showName();
    system("pause");
}
```

다중 상속 (Multiple Inheritance)

다중 상속

여러 개의 클래스로부터 멤버를 상속 받는 것을 말함.

한계

- 여러 개의 부모 클래스에 동일한 멤버가 존재할 가능성
- 하나의 클래스를 의도치 않게 여러 번 상속받을 가능성

// 다중상속

```
#include <iostream>
#include <string>

using namespace std;

class Person {
private:
    string name;
public:
    Person(string name) : name(name) {}
    string getName() {
        return name;
    }
    void showName() {
        cout << "나의이름: " << getName() << '\n';
    }
};
```

```
class Temp {
public:
    void showTemp() {
        cout << "임시 부모 클래스 \n";
    }
};

class Myname : Person, public Temp {
private:
    int Mynumber;
public:
    Myname(int Mynumber, string name) : Person(name) {
        this->Mynumber = Mynumber;
    }
    void show() {
        cout << "나의번호: " << Mynumber << '\n';
        cout << "나의이름: " << getName() << '\n';
    }
    //오버라이드
    void showName() {
        cout << "나의이름: " << getName() << '\n';
    }
};

int main(void) {
    Myname myname(1, "김하윤");
    myname.showName();
    myname.showTemp();
    system("pause");
}
```

C:\class\task\wx64\Debug\task.exe

나의이름: 김하윤
임시 부모 클래스
계속하려면 아무 키나 누르십시오 . . .

가상 함수 (Virtual Function)

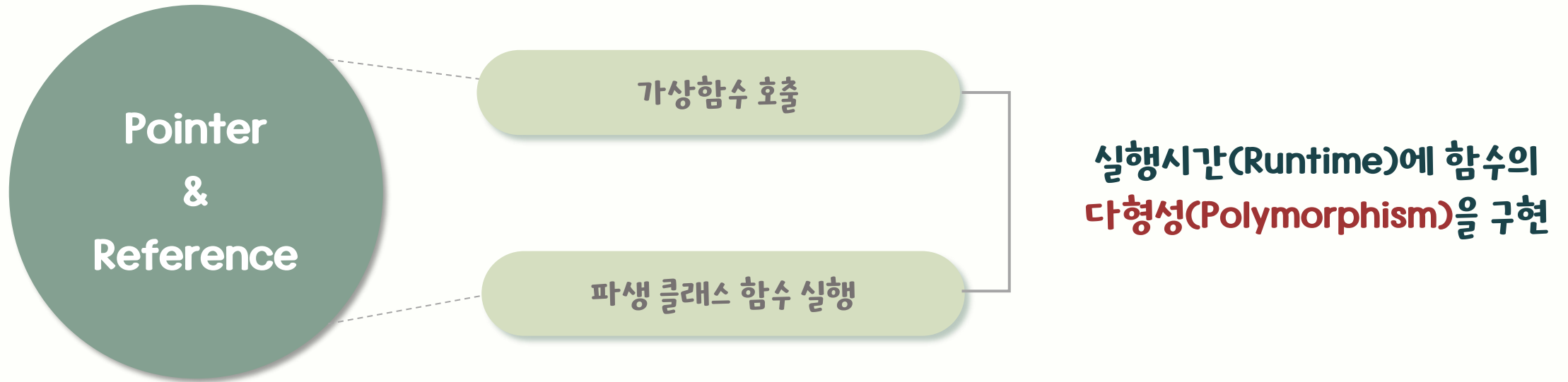
가상함수

기본 클래스 내에서 선언되어 파생 클래스에 의해 재정의되는 멤버 함수

➡ 자신을 호출하는 객체의 동적 타입에 따라 실제 호출할 함수가 결정

문법

`virtual` 멤버함수의 원형;



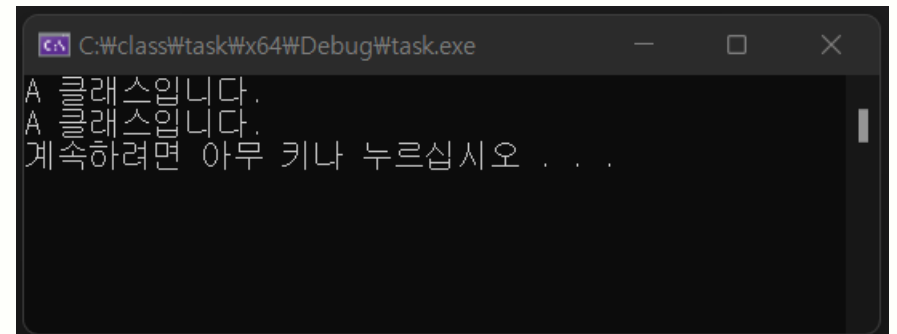
일반 함수의 정적 바인딩

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

class A
{
public:
    void show() { cout << "A 클래스입니다." << endl; }
};

class B : public A
{
    void show() { cout << "B 클래스입니다." << endl; }
};

int main(void)
{
    A* p;
    A a;
    B b;
    p = &a; p->show();
    p = &b; p->show(); // 여전히 A 클래스의 show() 함수를 호출합니다.
    system("pause");
}
```



```
C:\class\task\wx64\Debug\task.exe
A 클래스입니다.
A 클래스입니다.
계속하려면 아무 키나 누르십시오 . . .
```

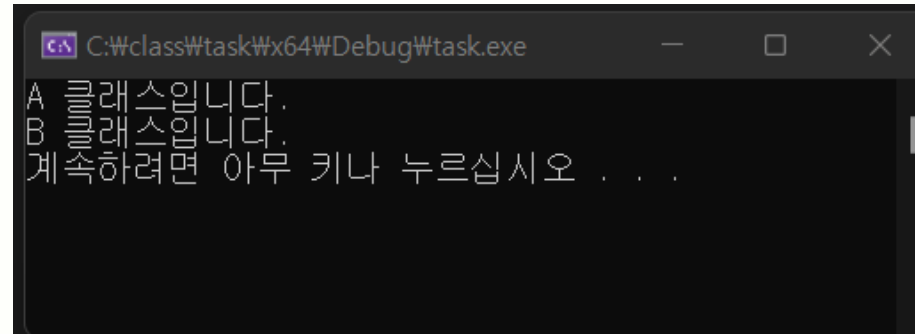
가상 함수의 동적 바인딩

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

class A
{
public:
    virtual void show() { cout << "A 클래스입니다." << endl; }
};

class B : public A
{
    virtual void show() { cout << "B 클래스입니다." << endl; }
};

int main(void)
{
    A* p;
    A a;
    B b;
    p = &a; p->show();
    p = &b; p->show(); // 여전히 A 클래스의 show() 함수를 호출합니다.
    system("pause");
}
```



C:\class\task\wx64\Debug\task.exe

A 클래스입니다.
B 클래스입니다.
계속하려면 아무 키나 누르십시오 . . .

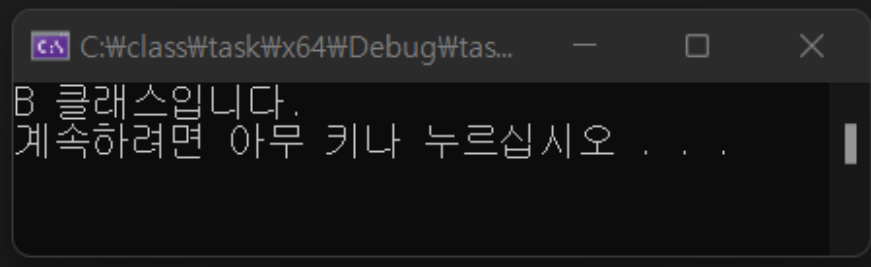
순수 가상 함수 (Pure Virtual Function)

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

class A
{
public:
    virtual void show() = 0 { }
};

class B : public A
{
    virtual void show() { cout << "B 클래스입니다." << endl; }
};

int main(void)
{
    A* p;
    B b;
    //p = &a; p->show();
    p = &b; p->show(); // 여전히 A 클래스의 show() 함수를 호출합니다.
    system("pause");
}
```



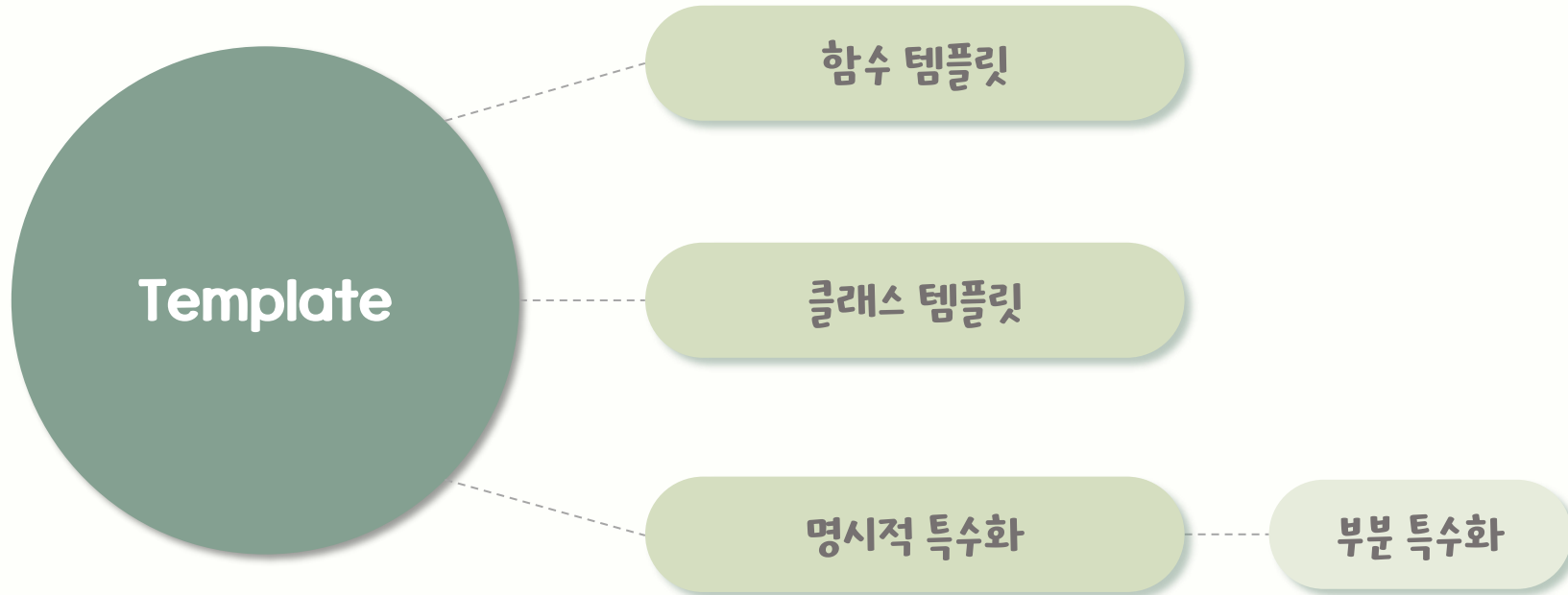
C:\class\task\wx64\Debug\tas...
B 클래스입니다.
계속하려면 아무 키나 누르십시오 . . .

템플릿 (Template)

템플릿

매개변수의 타입에 따라서 별도의 함수 및 클래스를 만들지 않고 다양한 타입에서 동작하는 단 하나의 객체를 정의할 수 있으며 소스코드의 재사용성을 극대화함.

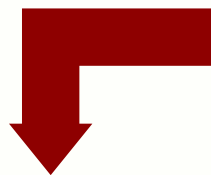
➡ C++은 일반화 프로그래밍이 가능한 언어이므로 템플릿을 이용해 일반화 프로그래밍 사용 가능



함수 템플릿 (Function Template)

문법

```
template <typename 타입이름>
함수의 원형
{
    // 클래스 멤버의 선언
}
```



```
#include <iostream>
#include <string>

using namespace std;

template <typename T>
void Swap(T& a, T& b);

int main(void)
{
    int c = 2, d = 3;
    cout << "c : " << c << ", d : " << d << endl;
    Swap(c, d);
    cout << "c : " << c << ", d : " << d << endl;

    string e = "KIM", f = "LEE";
    cout << "e : " << e << ", f : " << f << endl;
    Swap(e, f);
    cout << "e : " << e << ", f : " << f << endl;
    return 0;
}

template <typename T>
void Swap(T& a, T& b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}
```

Microsoft Visual Studio 디버그 콘솔

```
c : 2, d : 3
c : 3, d : 2
e : KIM, f : LEE
e : LEE, f : KIM
```

C:\#\class\#\task\#\x64\#\Debug\#\task.exe(프로세스 40264개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

클래스 템플릿 (Class Template)

문법

```
template <typename 타입이름>
class 클래스템플릿이름
{
    // 클래스 멤버의 선언
}
```



```
#include<iostream>
#include<string>

using namespace std;

template<typename T>
class Data {
private:
    T data;
public:
    Data(T data) : data(data) { }
    void setData(T data) { this->data = data; }
    T getData() { return data; }
};

int main(void) {
    Data<int> data1(1);
    Data<string> data2("김하윤");
    cout << data1.getData() << " " << data2.getData() << "\n";
    system("pause");
}
```

```
C:\class\task\wx64\Debug\task.exe
1:김하윤
계속하려면 아무 키나 누르십시오 . . .
```

명시적 특수화 (Explicit Specialization)

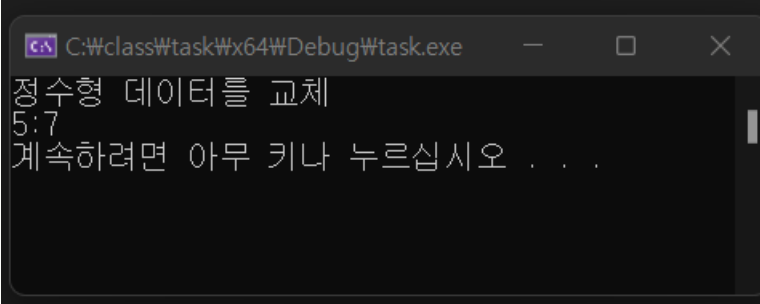
```
#include<iostream>
#include<string>

using namespace std;

template<typename T>
void change(T& a, T& b) {
    T temp;
    temp = a;
    a = b;
    b = temp;
}

template <> void change<int>(int& a, int& b)
{
    cout << "정수형 데이터를 교체\n";
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main(void) {
    int a = 7;
    int b = 5;
    change(a, b);
    cout << a << " : " << b << endl;
    system("pause");
}
```

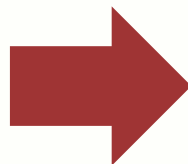


C:\class\Task\#x64\Debug\Task.exe

정수형 데이터를 교체
5:7
계속하려면 아무 키나 누르십시오 . . .

부분 특수화 (Partial Specialization) – 일반

```
template<typename T, int size>  
void print(StaticArray<T, size>& array)  
{  
    for (int count = 0; count < size; ++count)  
        cout << array[count] << ' ';  
    cout << endl;  
}
```



```
template<int size>  
void print(StaticArray<char, size>& array)  
{  
    for (int count = 0; count < size; ++count)  
        cout << array[count]; // 이곳이 달라짐  
    cout << endl;  
}
```

Microsoft Visual Studio 디버그 콘솔

```
1 2 3 4  
Hello, World ?
```

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

Microsoft Visual Studio 디버그 콘솔

```
1 2 3 4  
Hello, World!
```

C:\class\task\64\Debug\task.exe (프로세스 37160개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합

부분 특수화 (Partial Specialization) - 상속

```
#include <iostream>
#include <string>

using namespace std;

template <typename T, int size>
class StaticArray_BASE
{
private:
    T array_[size];

public:
    T* getArray() { return array_; }
    T& operator[](int index)
    {
        return array_[index];
    }

    void print()
    {
        for (int count = 0; count < size; ++count)
            cout << (*this)[count] << ' ';
        cout << endl;
    }
};

template <typename T, int size>
class StaticArray : public StaticArray_BASE<T, size>
{
};
```

```
template <int size>
class StaticArray<char, size> : public StaticArray_BASE<char, size>
{
public:
    void print()
    {
        for (int count = 0; count < size; ++count)
            cout << (*this)[count];
        cout << endl;
    }
};
```

```
int main()
{
    StaticArray<int, 4> int4;

    int4[0] = 1;
    int4[1] = 2;
    int4[2] = 3;
    int4[3] = 4;

    int4.print();

    StaticArray<char, 14> char14;

    strcpy_s(char14.getArray(), 14, "Hello, World!");

    char14.print();
}
```

Microsoft Visual Studio 디버그 콘솔

1 2 3 4
Hello, World!

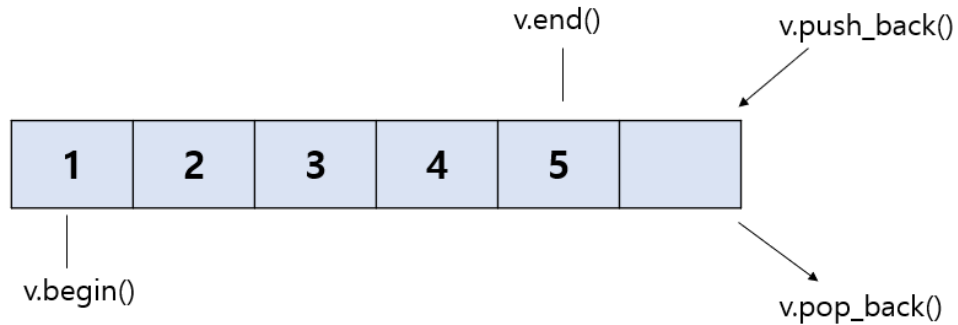
C:\#\class\task\#x64\Debug\task.exe (프로세스 37160개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정함

벡터 컨테이너 (Vector container)

벡터 컨테이너

벡터 컨테이너는 자동으로 메모리가 할당되는 배열이다.

모든 STL이 그렇듯이, Template 를 사용하기 때문에 데이터 타입을 마음대로 넣을 수 있다.



벡터의 사용

- `<vector>` 헤더파일을 추가해야 함.
- `using namespace std;`를 해주면 편리함.
- vector의 선언 : **`vector<[data type]> [변수이름]`**
ex) `vector<int>v;`

벡터의 생성자 & 연산자

```
vector<int> v;
```

비어 있는 vector v 를 생성

```
vector<int> v(x);
```

기본값(0)으로 초기화 된 x 개의 원소를 가지는 vector v 생성

```
vector<int> v(5,2);
```

2로 초기화된 5개의 원소를 가지는 vector v 생성

```
vector<int> v2(v1);
```

$v2$ 는 $v1$ vector를 복사해서 사용

If) `vector<int> v1;` , `vector<int> v2;` 가 있고, 내부에 인자들이 있을 때 ,

연산자 : "`==`" , "`!=`" , "`<`" , "`>`" , "`<=`" , "`>=`"로 대소비교가 가능

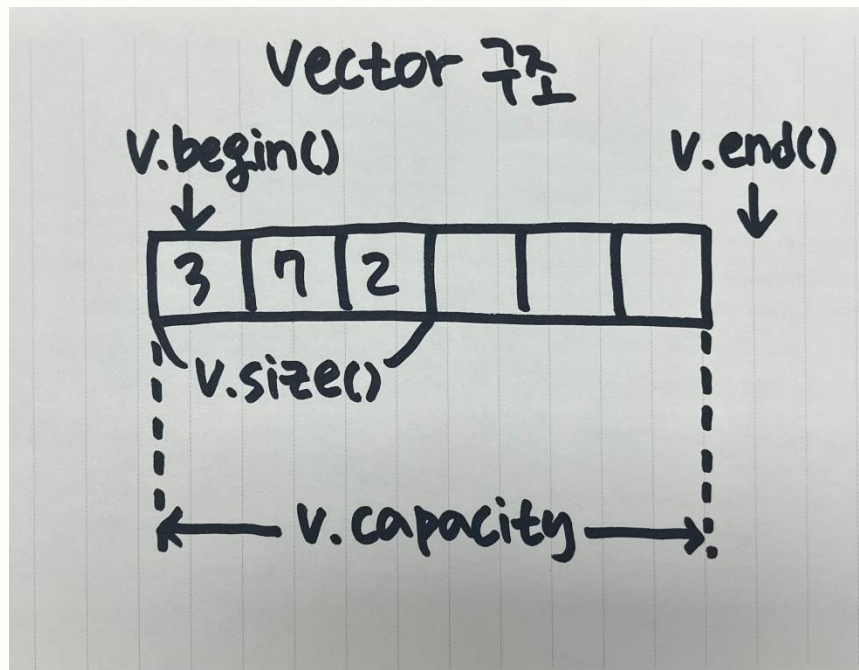
v.size(); & v.capacity();

v.size();

원소의 개수를 리턴

v.capacity();

할당된 공간의 크기를 리턴하고
공간 할당의 기준은 점점 커지면서 capacity 할당



v.size(); & v.capacity(); 에 대한 TEST

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main(void) {
    vector<int> v;

    cout << "[v[i],v.size(),v.capacity()]" << endl;
    for (int i = 0; i <= 64; i++)
    {
        v.push_back(i + 1);
        cout << "[" << v[i] << ", " << v.size() << ", " << v.capacity() << "]" << endl;
    }
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔 Microsoft Visual Studio 디버그 콘솔

[v[i],v.size(),v.capacity()]	[30,30,42]
[1,1,1]	[31,31,42]
[2,2,2]	[32,32,42]
[3,3,3]	[33,33,42]
[4,4,4]	[34,34,42]
[5,5,6]	[35,35,42]
[6,6,6]	[36,36,42]
[7,7,9]	[37,37,42]
[8,8,9]	[38,38,42]
[9,9,9]	[39,39,42]
[10,10,13]	[40,40,42]
[11,11,13]	[41,41,42]
[12,12,13]	[42,42,42]
[13,13,13]	[43,43,63]
[14,14,19]	[44,44,63]
[15,15,19]	[45,45,63]
[16,16,19]	[46,46,63]
[17,17,19]	[47,47,63]
[18,18,19]	[48,48,63]
[19,19,19]	[49,49,63]
[20,20,28]	[50,50,63]
[21,21,28]	[51,51,63]
[22,22,28]	[52,52,63]
[23,23,28]	[53,53,63]
[24,24,28]	[54,54,63]
[25,25,28]	[55,55,63]
[26,26,28]	[56,56,63]
[27,27,28]	[57,57,63]
[28,28,28]	[58,58,63]
[29,29,42]	[59,59,63]

벡터의 응용 – 중복된 원소 제거

```
#include<iostream>
#include<vector>
#include<algorithm>
```

unique 함수사용 .

```
using namespace std;
```

```
int main(void) {
    vector<int> v = { 1, 3, 2, 1, 8, 3, 5, 2, 6, 5, 8 };
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for (int i = 0; i < v.size(); i++) cout << v[i] << ' ';
}
```

```
#include<iostream>
#include<vector>
#include<algorithm>
```

resize 함수사용 .

```
using namespace std;
```

```
int main(void) {
    vector<int> v = { 1, 3, 2, 1, 8, 3, 5, 2, 6, 5, 8 };
    sort(v.begin(), v.end());
    vector<int>::iterator it;
    v.resize(unique(v.begin(), v.end()) - v.begin());
    for (int i = 0; i < v.size(); i++) cout << v[i] << ' ';
}
```



Microsoft Visual Studio 디버그 콘솔

```
1 2 3 5 6 8
C:\class\task\64\Debug\task.exe(프로세스 30728개)이(
(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] ->
] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도
이 창을 닫으려면 아무 키나 누르세요...
```



Microsoft Visual Studio 디버그 콘솔

```
1 2 3 5 6 8
C:\class\task\64\Debug\task.exe(프로세스 21920개)이(가)
종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵
션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]
를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

벡터의 응용 – Iterator 활용하여 벡터 출력

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v1;
    for (int i = 1; i <= 5; i++)
        v1.push_back(i);

    for (auto i = v1.begin(); i != v1.end(); ++i)
        cout << *i << " ";

    cout << "\n";
    for (auto ir = v1.rbegin(); ir != v1.rend(); ++ir)
        cout << *ir << " ";

    cout << "\n";
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";

    cout << "\n";
    for (int i = 0; i < v1.size(); i++)
        cout << v1.at(i) << " ";

    return 0;
}
```



Microsoft Visual Studio 디버그 콘솔

```
1 2 3 4 5
5 4 3 2 1
1 2 3 4 5
1 2 3 4 5
C:\class\task\64\Debug\task.exe(프로세스 31720개)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...
```

감사합니다.

T H A N K Y O U