

자료구조 & 알고리즘

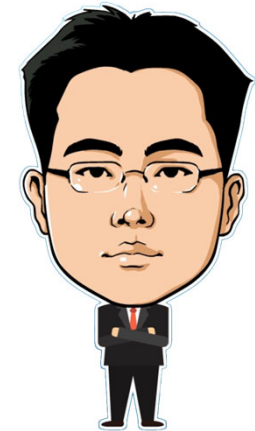
```
for(A;B;C)  
D;
```



C++ 프로그래밍
(C++ Programming)

Seo, Doo-Ok

Clickseo.com
clickseo@gmail.com



목 차



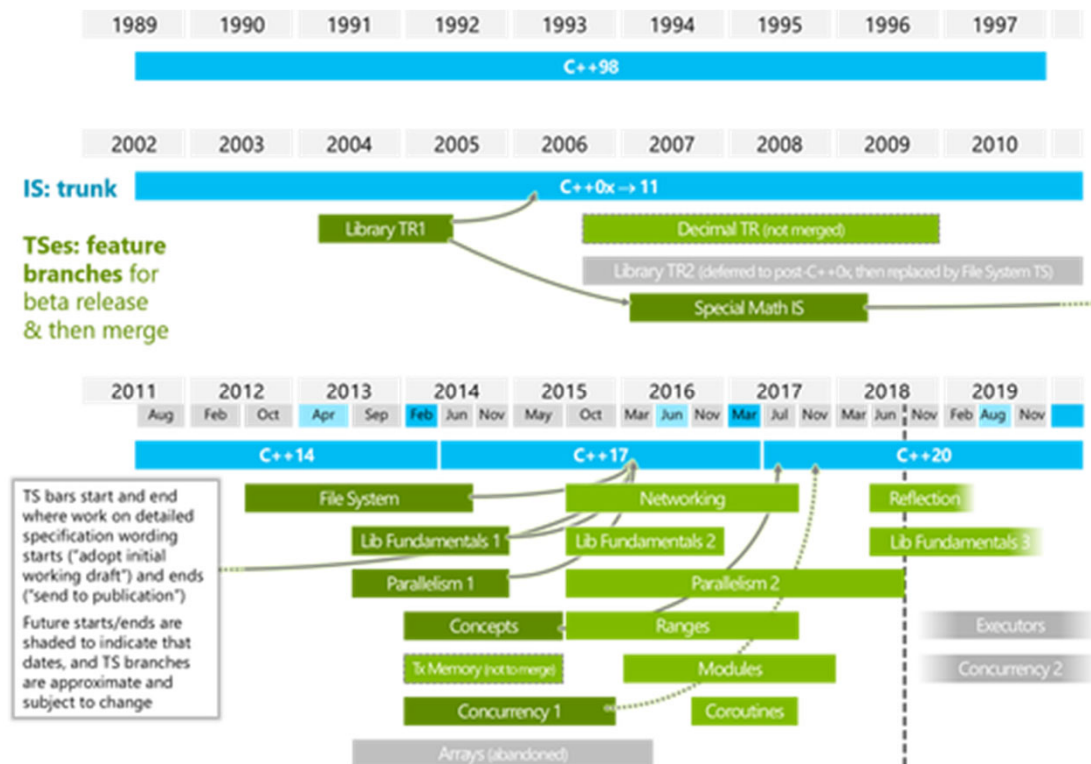
- C++ 프로그래밍 기초
- 객체지향 프로그래밍
- 템플릿과 STL



C++ 언어 개요

- C++ 언어 표준화

- 2018년 06월, ISO C++ standards meeting, "WG21 timeline"



[출처 : Herb Sutter, "Trip report : Summer ISO C++ standards meeting", 2018.]

C++ 프로그래밍 기초



- C++ 프로그래밍 기초
 - C++ 프로그램 구조
 - 배열, 문자열, 구조체
 - 함수, 네임스페이스, 참조
 - 동적 메모리 할당
- 객체지향 프로그래밍
- 템플릿과 STL



C++ 프로그래밍

C++ 프로그램 구조



C++ 프로그램 구조 (1/4)

- 새로운 형태의 자료형: **bool**

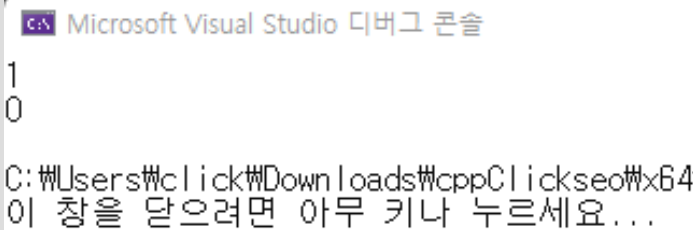
- bool형 변수의 상태는 **true**와 **false** 둘 중 하나가 될 수 있다.

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;

int main(void)
{
    std::cout << true << std::endl;
    std::cout << false << std::endl;

    return 0;
}
```



Microsoft Visual Studio 디버그 콘솔

```
1
0
```

C:\Users\click\Downloads\cppClickseo\x64
이 창을 닫으려면 아무 키나 누르세요...

C++ 프로그램 구조 (2/4)

● 변수(variable)

○ 변수 이름의 길이에 제한이 없다.

- C 에서 변수 이름의 길이 제한: 63번째 문자까지만 인식
- “지역 변수 선언의 위치 제한이 없다.”

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;

int main(void)
{
    int    i = 100;
    std::cout << i << std::endl;

    int    j = 200;
    std::cout << j << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

100
200

C:\Users\click\Downloads\cppClickseo\x64
이 창을 닫으려면 아무 키나 누르세요...

C++ 프로그램 구조 (3/4)

예제 0-1: 콘솔 입출력 -- cin, cout

```
#include <iostream>

// using std::cin;
// using std::cout;
// using std::endl;
// using namespace std;
```

```
int main(void)
{
    int    a, b, res;

    std::cout << "두 정수 입력: " ;
    std::cin  >> a >> b;

    res = a + b;

    std::cout << a << " + " << b << " = " << res << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

두 정수 입력: 10 20
10 + 20 = 30

C:\Users\click\Downloads\cppClickseo\64
이 창을 닫으려면 아무 키나 누르세요...

C++ 프로그램 구조 (4/4)

- 명시적 형 변환(explicit type conversion)

- cast 수식 연산자(cast expression operator)

- 임의로 어떤 형식에서 다른 형식으로 데이터를 변환시킨다.

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;
```

```
int main(void)
```

```
{
```

```
    int    i;
    double d = 3.14159;
```

```
    // i = (int)d;
```

```
    i = int(d);
```

// C++ 에서만 가능

```
    std::cout << "i: " << i << std::endl;
```

```
    std::cout << "d: " << d << std::endl;
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 디버그 콘솔

i: 3

d: 3.14159

C:\Users\click\Downloads\cppClickseo\x64\이 창을 닫으려면 아무 키나 누르세요...

C++ 프로그래밍 기초

배열, 문자열, 구조체



배열, 문자열, 구조체 (1/4)

- C 언어 스타일의 문자열: <cstring>

- C++ 에서 제공하는 문자열 조작 함수를 사용: <cstring>

```
#include <iostream>
#include <cstring>

// using std::cout;
// using std::endl;
// using namespace std;
```

```
int main(void)
{
    char    src[] = "Hi~ Clickseo!!!";
    char    dest[1024];
    int     len = (int)strlen(src);

    // strcpy(copy, str);
    strcpy_s(dest, sizeof(dest), src);

    std::cout << "길이 : " << len << std::endl;
    std::cout << "str : " << src << std::endl;
    std::cout << "copy: " << dest << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
길이 : 15
str : Hi~ Clickseo!!!
copy: Hi~ Clickseo!!!
```

```
C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...
```

배열, 문자열, 구조체 (2/4)

- C++ 언어 스타일의 문자열: <string>

- **string** : **basic_string** 클래스를 재정의한 형태

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

// using std::cout;
// using std::endl;
// using std::string;
// using namespace std;

int main(void)
{
    std::string s = "Hello World!!!!";

    std::cout << s << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

Hello World!!!!

C:\Users\click\Downloads\cppClickseo\64비
이 창을 닫으려면 아무 키나 누르세요...

문자열 복사: `str2 = str1`

문자열의 결합: `str1 + str2`

문자열의 비교 : `str1 == str2 / str1 != str2`

배열, 문자열, 구조체 (3/4)

- 구조체(Structure)

- C++에서는 구조체의 태그(tag)가 곧 자료형이다.

```
#include <iostream>

struct _score {
    char    name[12];
    int     kor, eng, math, tot;
    float   ave;
};

int main(void)
{
    // struct _score temp;           // C 언어 스타일
    _score  temp;                    // C++ 언어 스타일

    return 0;
}
```

배열, 문자열, 구조체 (4/4)

예제 0-2: 태그형 구조체

```
#include <iostream>

// using std::cin;
// using std::cout;
// using std::endl;
// using namespace std;

struct _score {
    char    name[12];
    int     kor, eng, math, tot;
    float   ave;
};

int main(void)
{
    _score  temp;

    std::cout << "이름: ";          std::cin >> temp.name;
    std::cout << "국어: ";          std::cin >> temp.kor;
    std::cout << "영어: ";          std::cin >> temp.eng;
    std::cout << "수학: ";          std::cin >> temp.math;

    temp.tot = temp.kor + temp.eng + temp.math;
    temp.ave = float(temp.tot) / 3;

    std::cout << "\n##### 성적 결과 출력 #####" << std::endl;
    std::cout << temp.name << " " << temp.kor << " " << temp.eng << " "
               << temp.math << " " << temp.tot << " " << temp.ave << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

이름: 서두옥
국어: 70
영어: 80
수학: 91

성적 결과 출력 #####
서두옥 70 80 91 241 80.3333

C:\Users\click\Downloads\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

C++ 프로그래밍 기초

함수



함수 (1/5)

- 함수 다중 정의(Function Overloading)

- C++ 언어에서 함수들이 동일한 이름을 사용할 수 있는 기능
 - 단, 인자의 종류(매개 변수의 자료형이나 개수)는 달라야 한다.

```
int      ADD(int, int);
double   ADD(double, double);

int main(void)
{
    ADD(10, 20);
    ADD(10.5, 20.5);

    return 0;
}

int      ADD(int a, int b) {
    return a + b;
}

double   ADD(double a, double b) {
    return a + b;
}
```


함수 (2/5)

● 디폴트 인자(Default Arguments)

○ 따로 값을 지정해주지 않은 경우에 선택하는 인자의 값

- 함수 호출 시 적당한 값을 모르는 경우에 사용
- 매번 함수를 호출할 때마다 똑같은 인자의 값을 적어주는 것을 피하는 용도로 사용
- 디폴트 인자의 제한: 디폴트 인자는 오른쪽 끝에 모여 있어야 한다.

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;

int    ADD(int, int = 0);

int main(void)
{
    std::cout << ADD(10) << std::endl;
    std::cout << ADD(10, 20) << std::endl;

    return 0;
}

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

함수 (3/5)

- 함수 다중 정의 vs. 디폴트 인자

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;

int      ADD(int a);
int      ADD(int a, int b = 0);

int main(void)
{
    // error C2668: 'ADD': 오버로드 된 함수에 대한 호출이 모호합니다.
    std::cout << ADD(10) << std::endl;

    return 0;
}

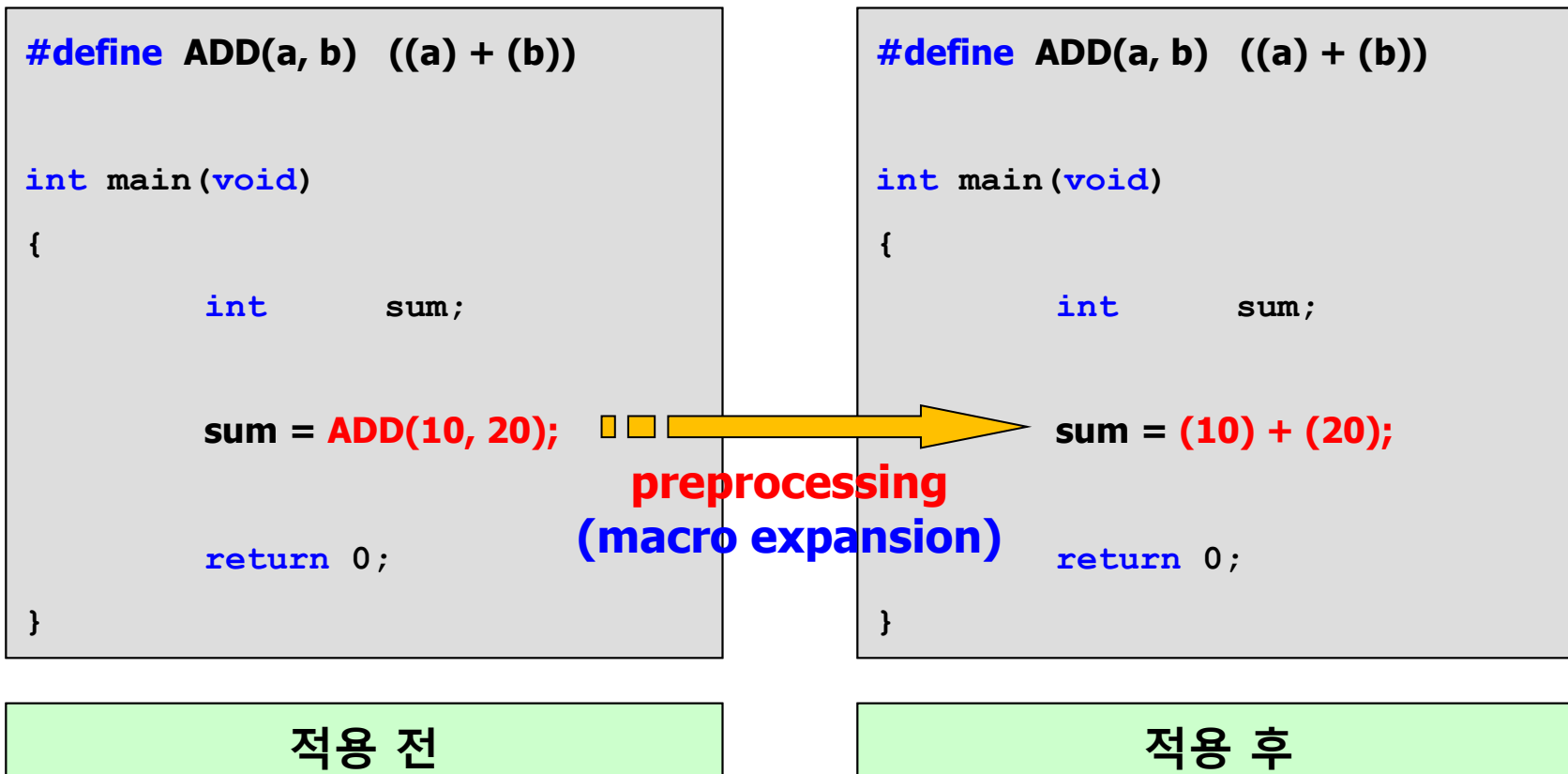
int      ADD(int a) {
    return a;
}

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

함수 (4/5)

- C 언어 스타일의 인-라인(in-line): 매크로 함수

- 프로그램을 컴파일 하기 전에 전처리에 의해 정의된 코드로 치환



함수 (5/5)

- C++ 언어 스타일의 인-라인(in-line)

- 인-라인 함수(in-line Function)

- 키워드 **inline**을 이용한 함수의 in-line화는 컴파일러에 의해 처리

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;
```

```
inline int ADD(int, int);
```

```
int main(void)
{
    int    sum;

    sum = ADD(10, 20);
    std::cout << "합계: " << sum << std::endl;

    return 0;
}
```

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

컴파일러에 의해 처리

C++ 프로그래밍 기초

네임스페이스



네임스페이스 (1/8)

- 네임스페이스의 등장 배경

- 같은 이름의 함수를 포함하면 컴파일 시 문제 발생

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;

void OUTPUT(void) {
    std::cout << "Hello World!!!" << std::endl;
    return;
}

// error C2084: 'void OUTPUT(void)' 함수에 이미 본문이 있습니다.
void OUTPUT(void) {
    std::cout << "Hi~ Clickseo" << std::endl;
    return;
}

int main(void)
{
    OUTPUT();

    return 0;
}
```

네임스페이스 (2/8)

- 네임스페이스(namespace)

- 특정 영역(공간)의 범위를 지정하고 이름을 붙여준 것

```
#include <iostream>

// using std::cout;
// using std::endl;
// using namespace std;
```

“이름 공간이 다르면
같은 이름의 변수나 함수의 선언이 허용된다.”

```
namespace A {
    void OUTPUT(void) {
        std::cout << "Hello World!!!" << std::endl;
        return;
    }
}
```

```
namespace B {
    void OUTPUT(void) {
        std::cout << "Hi~ Clickseo" << std::endl;
        return;
    }
}
```

```
int main(void)
{
    A::OUTPUT();
    B::OUTPUT();

    return 0;
}
```

범위 지정 연산자
(scope resolution operator)

네임스페이스 (3/8)

● 네임스페이스: 콘솔 입출력

```
#include <iostream>
int main(void)
{
    int    temp;

    std::cout << "정수 입력: ";
    std::cin  >> temp;
    std::cout << "temp: " << temp << std::endl;

    return 0;
}
```

```
namespace std {
    cout    ...
    cin     ...
    endl    ...
}
```

```
#include <iostream>
```

```
using std::cout;
using std::cin;
using std::endl;
```

```
using namespace std;
```

```
int main(void)
{
    int    temp;

    cout << "정수 입력: ";
    cin  >> temp;

    cout << "temp: " << temp << endl;

    return 0;
}
```


네임스페이스 (4/8)

- 네임스페이스: 범위 지정연산자

- 지역변수와 전역변수

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

int      temp;    // 전역변수

int main(void)
{
    int      temp = 10;    // 지역변수

    // 전역변수: temp
    ::temp++;

    cout << "지역변수: " << temp << endl;
    cout << "전역변수: " << ::temp << endl;

    return 0;
}
```

네임스페이스 (5/8)

- 네임스페이스: 별칭(Alias)

- 네임스페이스 별칭 부여

- 네임스페이스의 이름이 너무 긴 경우 간단한 별칭 사용하여 단순화하여 사용.

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

namespace Clickseo_namespace_data_temp {
    int    temp;
}

namespace Click = Clickseo_namespace_data_temp;

int main(void)
{
    cout << "temp : " << Click::temp << endl;

    return 0;
}
```

네임스페이스 (6/8)

- 네임스페이스: 중첩 네임스페이스

- 중첩된 네임스페이스

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

namespace Clickseo {
    namespace TEMP1 {
        int a = 10;
    }
    namespace TEMP2 {
        int a = 20;
    }
}

int main(void)
{
    cout << "Clickseo::TEMP1::a : " << Clickseo::TEMP1::a << endl;
    cout << "Clickseo::TEMP2::a : " << Clickseo::TEMP2::a << endl;

    return 0;
}
```

네임스페이스 (7/8)

- 네임스페이스: 이름 없는 네임스페이스

- 다른 파일에서 접근 제한

- static 키워드를 사용한 전역 변수나 함수를 정의한 경우와 동일한 효과

1.cpp

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

namespace {
    int temp = 10;
}

void OUTPUT(void)
{
    cout << "temp: " << temp << endl;

    return;
}
```

2.cpp

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

extern void OUTPUT(void);
extern int temp;

int main(void)
{
    cout << "temp: " << temp << endl;
    OUTPUT();

    return 0;
}
```

빌드 시작...

1>----- 빌드 시작: 프로젝트: cppClickseo, 구성: Debug x64 -----

1>2.obj : error LNK2001: 확인할 수 없는 외부 기호 "int temp" (?temp@@3HA)

1>C:\Users\Wclick\Downloads\WcppClickseo\W64\Debug\WcppClickseo.exe : fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.

1>"cppClickseo.vcxproj" 프로젝트를 빌드했습니다. - 실패

===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====

네임스페이스 (8/8)

main.cpp

```
namespace A { void OUTPUT(void); }
namespace B { void OUTPUT(void); }

int main(void)
{
    A::OUTPUT();
    B::OUTPUT();
    return 0;
}
```

분할 컴파일

1.cpp

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

namespace A {
    void OUTPUT(void) {
        cout << "Hello World!!!" << endl;
        return;
    }
}
```

2.cpp

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

namespace B {
    void OUTPUT(void) {
        cout << "Hi~ Clickseo" << endl;
        return;
    }
}
```

C++ 프로그래밍 기초

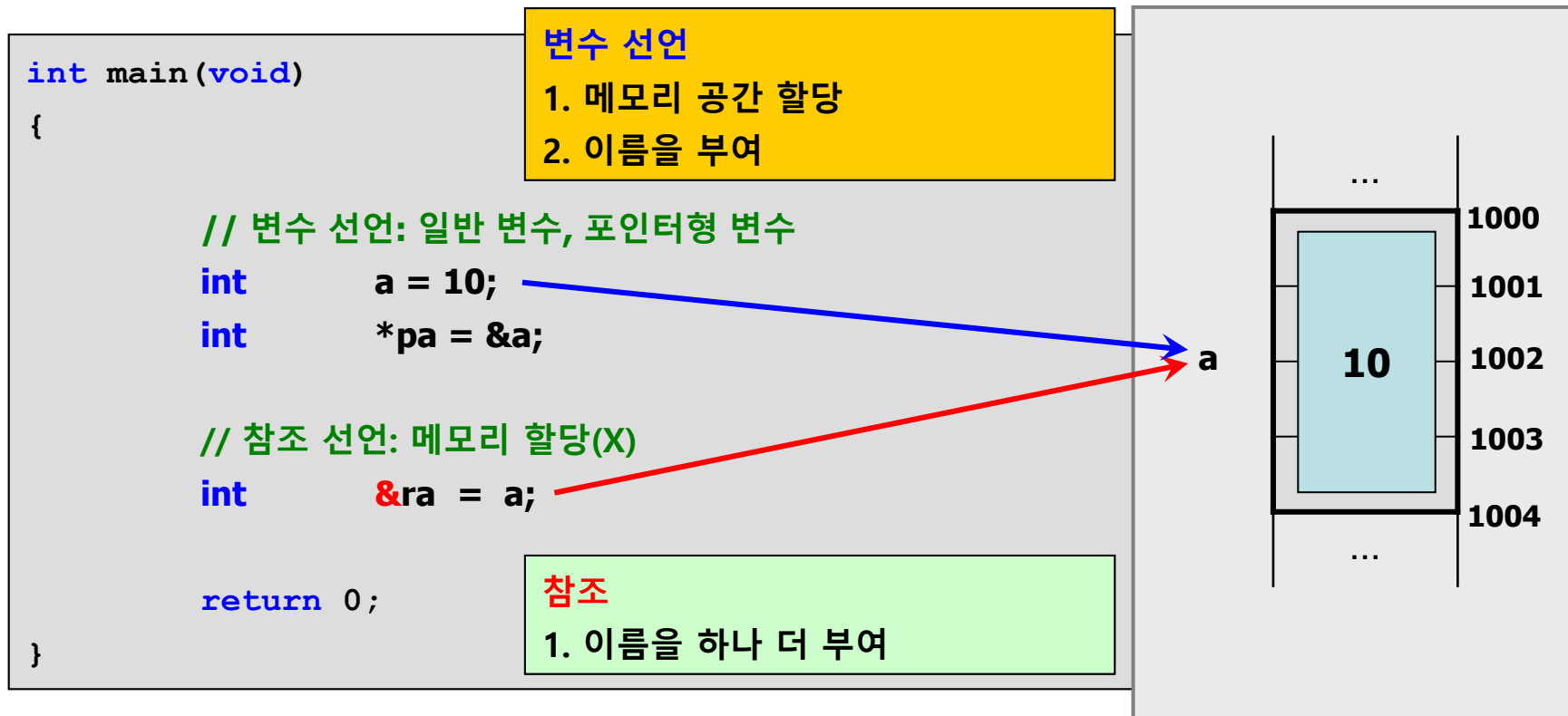
참조



참조 (1/5)

● 참조(Reference)

- 이름이 존재하는 메모리 공간에 하나의 이름을 더 부여하는 행위
 - 참조는 만드는 순간 초기화 되어야 한다(단, 상수로 초기화 할 수는 없다).



참조 (2/5)

예제 0-3: 참조(reference)의 이해 -- 일반 변수와 참조

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

int main(void)
{
    int    a = 10;
    int    &ra = a;

    a++;
    cout << "a   : " << a << endl;
    cout << "ra  : " << ra << endl;

    ra++;
    cout << "a   : " << a << endl;
    cout << "ra  : " << ra << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
a : 11
ra : 11
a : 12
ra : 12
```

C:\Users\click\Downloads\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

참조 (3/5)

● 참조에 의한 호출(Call by reference)

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

```
void SWAP(int &, int &);
```

```
int main(void)
{
```

```
    int    a = 10, b = 20;
```

```
    cout << "a: " << a << " , b: " << b << endl;
    SWAP(a, b);
    cout << "a: " << a << " , b: " << b << endl;
```

```
    return 0;
```

```
}
```

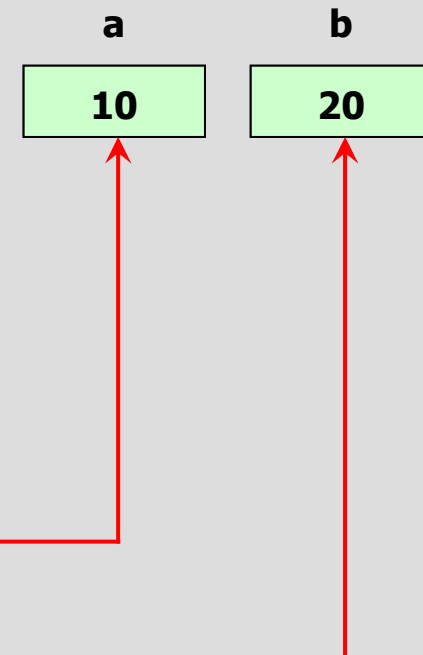
```
void SWAP(int &ra, int &rb) {
```

```
    int    temp;
```

```
    temp = ra;
    ra = rb;
    rb = temp;
```

```
    return;
```

```
}
```



참조 (4/5)

예제 0-4: 부담스러운 값에 의한 호출(Call-by-value)

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

struct _score {
    char    name[12];
    int     kor, eng, math, tot;
    float   ave;
};

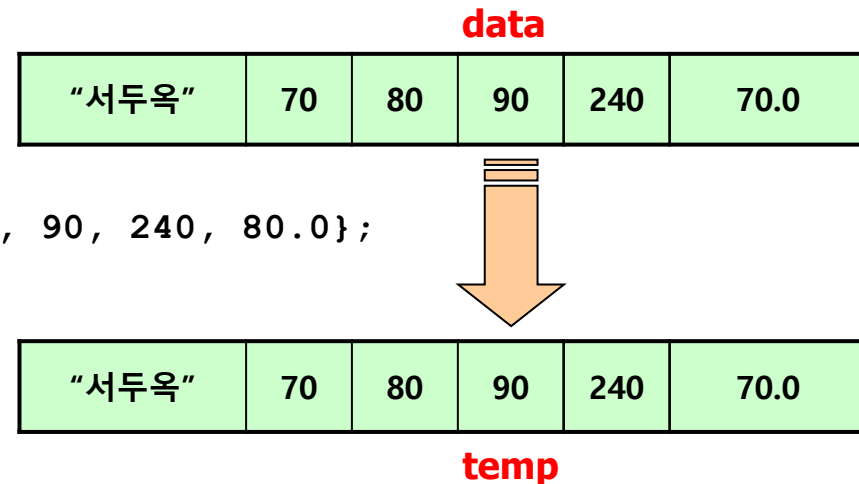
void OUTPUT(_score);

int main(void)
{
    _score data = {"서두옥", 70, 80, 90, 240, 80.0};
    OUTPUT(data);

    return 0;
}

void OUTPUT(_score temp) {
    cout << "이름: " << temp.name << endl;
    cout << "국어: " << temp.kor << endl;
    cout << "영어: " << temp.eng << endl;
    cout << "수학: " << temp.math << endl;
    cout << "총점: " << temp.tot << endl;
    cout << "평균: " << temp.ave << endl;

    return;
}
```



참조 (5/5)

예제 0-5: 참조에 의한 호출(Call-by-reference)

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

struct _score {
    char    name[12];
    int     kor, eng, math, tot;
    float   ave;
};

void OUTPUT(const _score &);

int main(void)
{
    _score data = {"서두옥", 70, 80, 90, 240, 80.0};
    OUTPUT(data);
    return 0;
}

void OUTPUT(const _score &temp) {
    cout << "이름: " << temp.name << endl;
    cout << "국어: " << temp.kor << endl;
    cout << "영어: " << temp.eng << endl;
    cout << "수학: " << temp.math << endl;
    cout << "총점: " << temp.tot << endl;
    cout << "평균: " << temp.ave << endl;

    return;
}
```

"서두옥"	70	80	90	240	70.0
-------	----	----	----	-----	------

data

temp

C++ 프로그래밍 기초

동적 메모리 할당



동적 메모리 할당 (1/2)

- 동적 메모리 할당: new, delete

- new : 동적 메모리 할당

- 메모리 할당 실패 시 NULL 포인터 반환

<code>int *p = new int;</code>	// 정수 하나를 저장할 메모리 할당
<code>int *pArr = new int[arrMAXSIZE];</code>	// 정수를 size 개수만큼 저장할 메모리 할당

- delete : 동적 메모리 해제

<code>delete p;</code>	// 할당된 메모리 공간 해제
<code>delete []pArr;</code>	// 할당된 메모리 공간이 배열일 경우

- C 언어 스타일의 동적 메모리 할당

- 표준 라이브러리 헤더 파일: **<stdlib.h>**
 - 메모리 할당: **malloc**, **calloc**, **realloc**
 - 메모리 해제: **free**

동적 메모리 할당 (2/2)

예제 0-6: 동적 메모리 할당 -- C++

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

int main(void)
{
    int    size;

    cout << "입력 할 학생 수: ";
    cin >> size;

    // 동적 메모리 할당
    int* pArr = new int[size];
    if(pArr == NULL) {
        cout << "메모리 할당 실패!!!" << endl;
        return -1;
    }

    cout << "\n ### 데이터 입력 ### " << endl;
    for(int i=0; i<size; i++) {
        cout << i + 1 << " : ";
        cin >> *(pArr + i);
    }

    cout << "\n ### 결과 출력 ### " << endl;
    for(int i=0; i<size; i++)
        cout << i << " : " << *(pArr + i) << endl;

    // 동적 메모리 해제
    delete []pArr;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

입력 할 학생 수: 3

데이터 입력

1 : 11

2 : 22

3 : 33

결과 출력

0 : 11

1 : 22

2 : 33

C:\Users\click\Downloads\cppClickseo\64\ 이 창을 닫으려면 아무 키나 누르세요...

객체지향 프로그래밍



- C++ 프로그래밍 기초
- 객체지향 프로그래밍
 - 클래스와 데이터 추상화
 - 연산자 다중 정의
 - 상속과 다형성
 - 예외 처리
- 템플릿과 STL

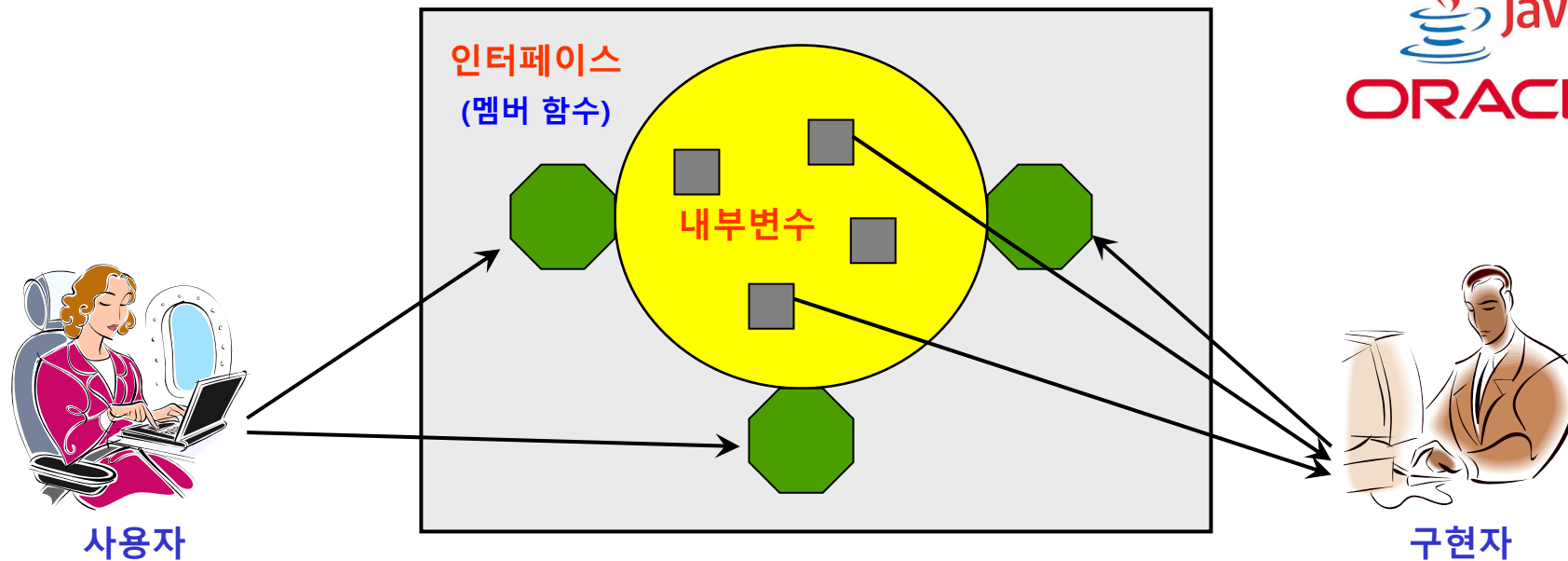
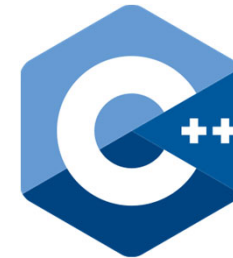


객체지향 프로그래밍 (1/2)

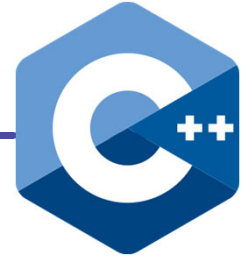
● 객체지향 프로그래밍(Object-Oriented Programming)

○ 객체(Object)들의 모임

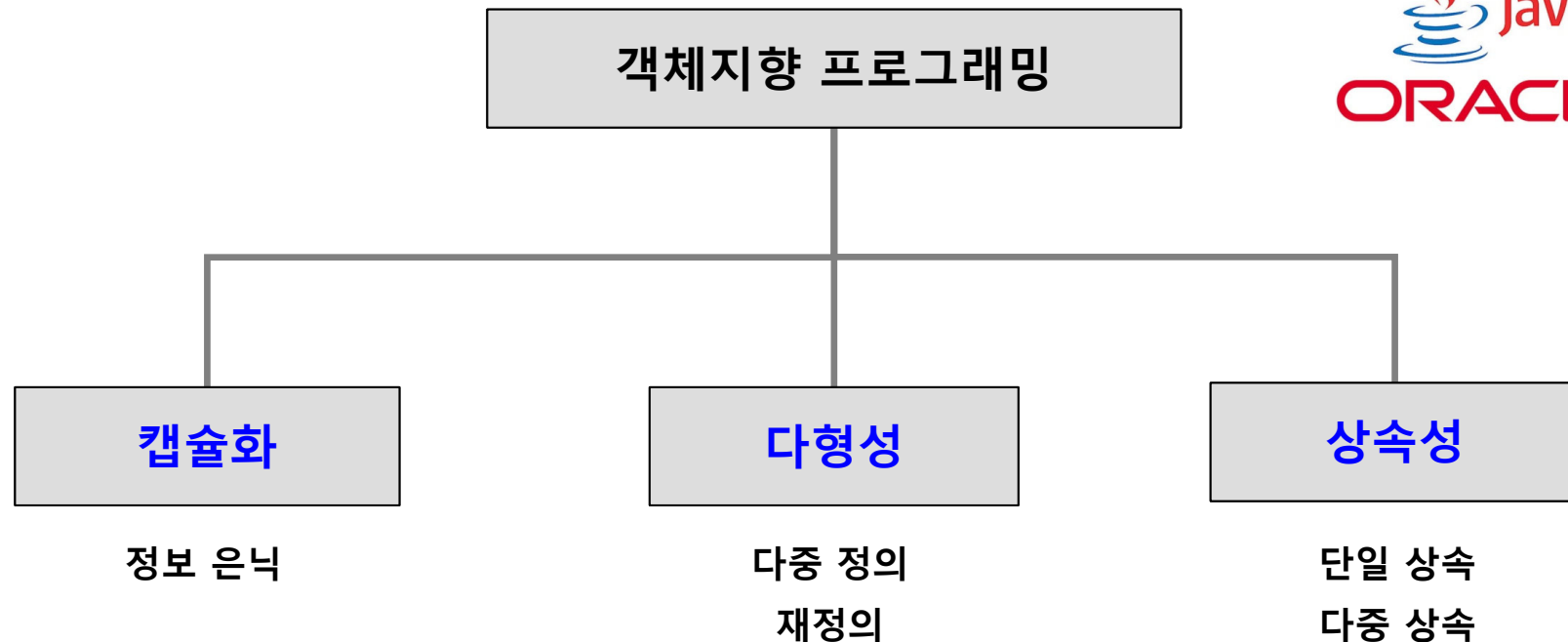
- 구성 요소: 클래스, 객체, 메소드, 메시지
- 특징: 캡슐화, 추상화, 다형성, 상속, 인스턴스 등
- C++, JAVA, C# 등



객체지향 프로그래밍 (2/2)



- 객체지향 프로그래밍의 3가지 특징



객체지향 프로그래밍

클래스와 데이터 추상화
: 클래스와 객체



클래스와 객체 (1/6)

● 클래스(Class)

○ 같은 목적을 가진 함수 변수들의 집합체(새로운 자료형)

- 변수 : 애트리뷰트(attribute)
- 함수 : 메소드(method)

// 클래스 정의

```
class Point {  
    // 멤버 변수  
    int x;  
    int y;  
    // 멤버 함수  
    void OUTPUT(void) {  
        cout << "x: " << x << ", y: " << y << endl;  
    }  
};
```

Point
int x; int y;
void OUTPUT();

클래스 = 멤버 변수 + 멤버 함수

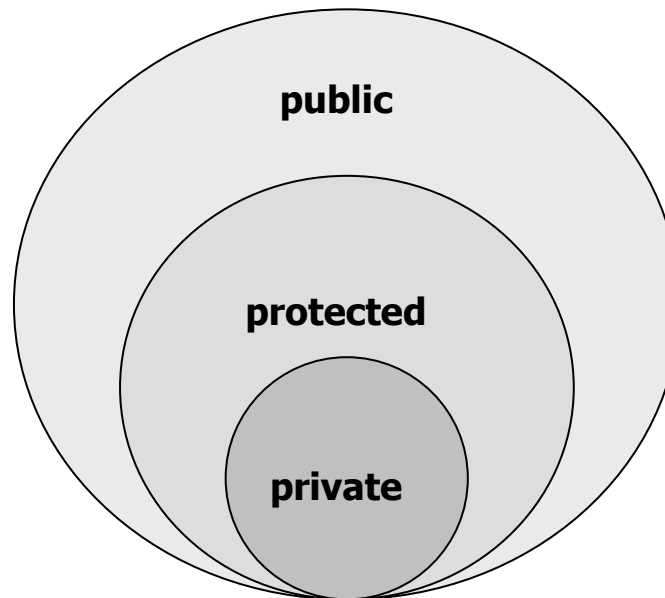
○ 객체(Object)

- 클래스를 이용해서 정의된 자료형의 변수의 표현(완전한 대상체)
- 인스턴스화(instantiation) : 클래스를 기반으로 객체를 생성하는 것

클래스와 객체 (2/6)

● 클래스 멤버 접근

- 클래스 내의 멤버 변수 또는 멤버 함수의 접근 권한 부여
 - **public** : 모든 클래스에서 접근 허용(외부 접근)
 - **protected** : 클래스와 상속받은 클래스의 접근 허용
 - **private** : 동일한 클래스의 접근만 허용(내부 접근)



클래스와 객체 (3/6)

예제 0-7: 클래스와 객체 -- 클래스 멤버 접근

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

// 클래스 정의
class Point {
public:
    // 멤버 변수
    int x;
    int y;
    void OUTPUT(void);    // 멤버 함수
};

// 멤버 함수: 클래스 외부 정의
void Point::OUTPUT(void) {
    cout << "x: " << x << ", y: " << y << endl;
}

int main(void)
{
    Point a;
    a.x = 10;
    a.y = 20;
    a.OUTPUT();

    // Point b(a);
    Point b = a;
    b.OUTPUT();

    return 0;
}
```

Point
int x; int y;
void OUTPUT();

Microsoft Visual Studio 디버그 콘솔

x: 10, y: 20
x: 10, y: 20

C:\Users\click\OneDrive\문서\cpp\Clickseo\64\
이 창을 닫으려면 아무 키나 누르세요...

클래스와 객체 (4/6)

- 좋은 클래스의 설계

- 정보은닉(Information Hiding)

- 모든 멤버 변수를 **private**으로 선언!!!
 - 객체의 외부에서 객체의 멤버 변수에 직접 접근하지 못하게 하는 것.
 - 오직 객체의 멤버 함수를 통하여 접근하도록 하는 방법

- 캡슐화(Encapsulation)

- 관련 있는 데이터와 함수를 하나의 단위로 묶는다.

```
class Point {  
    int x;  
    int y;  
public:  
    void OUTPUT(void);  
};
```

멤버 변수: private

```
void Point::OUTPUT(void) {  
    cout << "x: " << x << ", y: " << y << endl;  
}
```

클래스와 객체 (5/6)

- **friend 선언:** 클래스

- 다른 클래스에서 `private`으로 선언된 영역의 접근 허용
 - 단, friend 선언은 단방향성을 지닌다.

```
class Count {  
    int i;  
    friend class fCount;  
};  
  
class fCount {  
public:  
    void setCount(Count& r, int num) {  
        r.i = num;  
    }  
};
```

클래스와 객체 (6/6)

- **friend 선언: 전역 함수**

- friend 선언을 통해서 private으로 선언된 멤버 변수의 접근 허용

```
class Count {  
    int    i;  
public:  
    Count() { i = 0; }  
    void    showData(void) {  
        cout << "i: " << i << endl;  
    }  
    friend void    setCount(Count&, int);  
};  
  
void    setCount(Count &r, int num) {           // 전역 함수  
    r.i = num;  
}
```


객체지향 프로그래밍

클래스와 데이터 추상화
: 생성자와 소멸자



생성자와 소멸자 (1/7)

- **생성자(Constructor)**

- 객체의 생성과 동시에 호출되는 함수

- 클래스의 이름과 동일한 이름의 함수

- 반환하지도 않고, 반환되는 자료형도 선언되지 않는다.
 - 생성자를 하나도 정의하지 않으면, **디폴트(default) 생성자**가 자동 삽입된다.

```
class Point
{
    int    x;
    int    y;
public:
    Point() {}           // default 생성자
};
```

생성자와 소멸자 (2/7)

예제 0-8: 생성자와 함수 다중 정의

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

class Point {
    int _x;
    int _y;
public:
    Point() {}; // default 생성자
    Point(int x, int y) { // 생성자 함수
        _x = x;
        _y = y;
    }
    void showData(void) {
        cout << "x: " << _x << ", y: " << _y << endl;
    }
};
```

// 생성자도 함수이므로 함수의 특징을 그대로 지닌다.

```
int main(void)
{
    Point a;
    Point b(10, 20);

    a.showData();
    b.showData();

    return 0;
}
```

Point
int _x; int _y;
Point(); Point(int, int); void ShowData();

```
C:\Microsoft Visual Studio 디버그 콘솔
x: -858993460, y: -858993460
x: 10, y: 20
```

```
C:\Users\click\OneDrive\문서\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...
```

생성자와 소멸자 (3/7)

예제 0-9: 생성자와 디폴트 매개 변수

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

class Point {
public:
    int _x, _y;
    Point(int x = 0, int y = 0) {
        _x = x;
        _y = y;
    }
    void showData(void) {
        cout << "x: " << _x << ", y: " << _y << endl;
    }
};

int main(void)
{
    // Point a(0, 0);
    Point a;
    a.showData();

    Point b(10, 20);
    b.showData();

    return 0;
}
```

Point
int _x; int _y;
Point(); Point(int, int); void ShowData();

Microsoft Visual Studio 디버그 콘솔

x: 0, y: 0
x: 10, y: 20

C:\Users\click\OneDrive\문서\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

생성자와 소멸자 (4/7)

● 소멸자(Destructor)

- 객체 소멸 시 자동적으로 호출되는 함수
 - 객체 소멸 시 다양한 형태의 정리 작업 필요 시...
- 클래스 이름 앞에 ~가 붙은 형태의 함수
 - 함수 다중 정의와 디폴트 매개변수 불가!!!
 - 매개 변수와 반환되는 자료형은 선언되지 않는다.
 - 소멸자를 하나도 정의하지 않으면, 디폴트(default) 소멸자가 자동 삽입된다.

```
class Point {  
    int    x;  
    int    y;  
public:  
    ~Point() {};    // default 소멸자  
};
```

생성자와 소멸자 (5/7)

● 복사 생성자

- 자기 자신과 같은 형태의 객체를 인자로 받을 수 있는 생성자
 - 디폴트 복사 생성자: 자동으로 삽입되는 복사 생성자
 - 두 객체의 멤버 변수와 멤버 변수를 복사

```
class Point {  
    int    _x;  
    int    _y;  
public:  
    Point() { _x = _y = 0; };  
    Point(int x, int y = 0) {  
        _x = x;  
        _y = y;  
    }  
    Point(const Point& p)      {           // default 복사 생성자  
        _x = p.x;  
        _y = p.y;  
    }  
};
```

생성자와 소멸자 (6/7)

- 얕은 복사(shallow copy)

- 디폴트 복사 생성자의 문제점

```
class Person {  
    int    _id;  
    char*  _name;  
  
public:  
    Person(int id, char* name);  
    ~Person(void);  
    void    showData(void) ;  
};
```

```
Person    a(202255001, "홍길동");  
Person    b = a;  
Person    c(a);
```

```
// 디폴트 복사 생성자  
Person(const Person& p) {  
    _id = p.id;  
    _name = p.name;  
}
```

생성자와 소멸자 (7/7)

- 깊은 복사(Deep copy)

- 직접 복사 생성자를 제공

- 생성자 내에서 동적 할당을 하면, 반드시 제공되어야 한다.

```
class Person {  
    int    _id;  
    char   *_name;  
  
public:  
    Person(int id, char *name);  
    Person(const Person& p);  
    ~Person(void);  
    void    ShowData(void) ;  
};
```

```
Person::Person(const Person& p) {  
    _name = new char[strlen(p._name)+1];  
    strcpy(_name, p._name);  
}
```


객체지향 프로그래밍

클래스와 데이터 추상화
: 클래스와 포인터, 배열



클래스와 포인터, 배열 (1/6)

● 객체의 포인터

- 객체를 가리키는(참조하는) 용도로 사용되는 포인터

```
class Point {  
    int    x;  
    int    y;  
public:  
    void OUTPUT(void) { cout << "x: " << x << ", y: " << y << endl; }  
};  
  
int main(void)  
{  
    Point    a;  
    Point    *p = &a;  
  
    a.OUTPUT();  
  
    (*p).OUTPUT();  
    p->OUTPUT();  
  
    return 0;  
}
```

Point
int x; int y;
void OUTPUT();

클래스와 포인터, 배열 (2/6)

● 자기 참조 포인터: this

○ this는 자기 자신을 가리키는(참조하는) 용도로 사용되는 포인터

- 멤버 함수 내에서 this라는 이름의 포인터를 사용

```
class Point
{
    int x;
    int y;
public:
    Point *getThis(void) {
        return this;
    }
};
```

Point
int x; int y;
Point *getThis();

```
int main(void)
{
    Point *p = new Point();           // Point *p = new Point;

    cout << "p : " << p << endl;
    cout << "this: " << p->getThis() << endl;

    return 0;
}
```

클래스와 포인터, 배열 (3/6)

- 자기 참조 포인터: 멤버 변수

```
class Point {  
    int    x;  
    int    y;  
  
public:  
    Point() { x = y = 0; };  
    Point(int x, int y = 0) {  
        this->x = x;  
        this->y = y;  
    }  
    Point(const Point& p) {  
        this->x = p.x;  
        this->y = p.y;  
    }  
};
```

“this 포인터로
멤버 변수의 이름 충돌 해결!!!”

Point
int x; int y;
Poin(); Point(int, int = 0); Point(const Point &);

클래스와 포인터, 배열 (4/6)

● 객체의 배열과 포인터 배열

- **객체의 배열:** 클래스도 자료형의 한 종류이기 때문에 배열을 생성할 수 있다.
 - 객체의 배열을 정의할 때 각 객체들은 디폴트 생성자로 초기화된다.
- **객체의 포인터 배열:** 객체를 가리키는(참조하는) 용도로 사용되는 포인터 배열

```
class Point {  
    int    x;  
    int    y;  
public:  
    Point() { x = y = 0; };  
    Point(int x, int y = 0) {  
        this->x = x;  
        this->y = y;  
    }  
    Point(const Point& p)    {  
        this->x = p.x;  
        this->y = p.y;  
    }  
};
```

// 객체의 배열

```
Point    pArr[3];  
pArr[0];  
pArr[1];  
pArr[2];
```

// 객체의 포인터 배열

```
Point*    pArr[3];  
arr[0] = new Point();  
arr[1] = new Point(10, 20);  
arr[2] = new Point(*arr[1]);
```

클래스와 포인터, 배열 (5/6)

예제 0-10: 객체의 포인터 배열

(1/2)

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

class Point {
    int    x;
    int    y;
public:
    Point();
    Point(int, int = 0);
    Point(const Point&);
    void    ShowData(void) ;
};

Point::Point() { x = y = 0; };
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}
Point::Point(const Point& p) {
    this->x = p.x;
    this->y = p.y;
}
void    Point::showData(void) {
    cout << "x: " << x << ", y: " << y << endl;
}
```

Point
int x; int y;
Poin(); Point(int, int = 0); Point(const Point &); void showData();

클래스와 포인터, 배열 (6/6)

예제 0-10: 객체의 포인터 배열

(2/2)

```
int main(void)
{
    Point* pArr[3];

    pArr[0] = new Point();
    pArr[1] = new Point(10, 20);
    pArr[2] = new Point(*pArr[1]);

    for(int i = 0; i < 3; i++)
        pArr[i]->showData();

    for(int i = 0; i < 3; i++)
        delete pArr[i];

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
x : 0, y : 0
x : 10, y : 20
x : 10, y : 20
```

C:\Users\click\OneDrive\문서\cppClickseo\64\ 이 창을 닫으려면 아무 키나 누르세요...

객체지향 프로그래밍

연산자 다중 정의



연산자 다중 정의 (1/2)

- 연산자 다중 정의 (Operator Overloading)
 - operator와 연산자 기호를 통해 연산자의 기능을 다중 정의
 - 클래스를 C++의 기본 자료형과 같이 다룰 수 있는 방법

```
operator+() {  
    // 주어진 연산자의 기능 정의  
};
```



연산자 다중 정의 (2/2)

● 문자열과 연산자 다중 정의

○ C++ 언어 스타일의 문자열: <string>

- **string** : **basic_string** 클래스를 재정의한 형태

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

// using std::cout;
// using std::endl;
// using std::string;
// using namespace std;

int main(void)
{
    std::string s = "Hello World!!!!";
    std::cout << s << std::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

Hello World!!!!

C:\Users\click\Downloads\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

문자열 복사: `str2 = str1`

문자열의 결합: `str1 + str2`

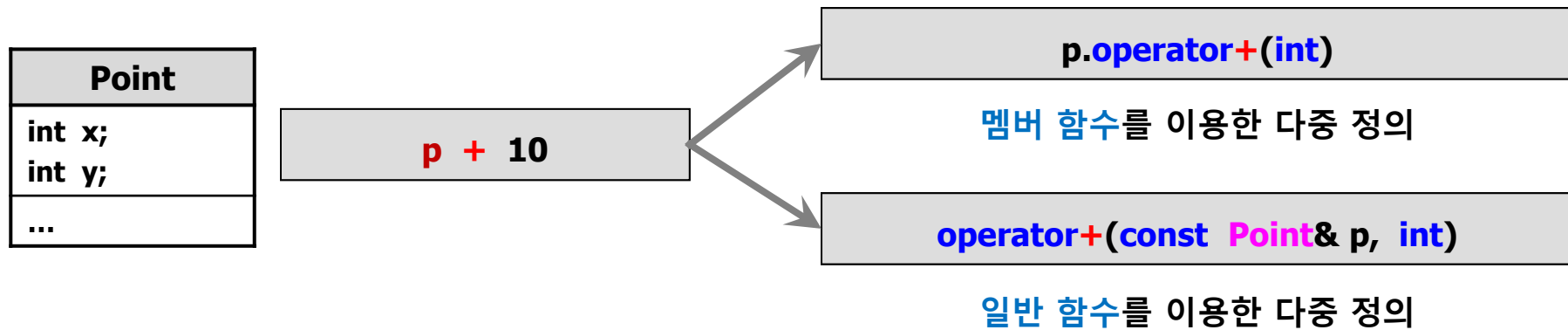
문자열의 비교 : `str1 == str2 / str1 != str2`

연산자 다중 정의: 이항 연산자 (1/3)

- 연산자 다중 정의: 이항 연산자

- 이항 연산자 다중 정의

- 이항 연산자(Binary Operator): 두 개의 피연산자에 대한 연산을 수행하는 연산자
 - 대표적인 이항 연산자: +, -, *, /, %



- 이항 연산자의 교환 법칙

$$\boxed{p + 10} \quad == \quad \boxed{10 + p}$$

연산자 다중 정의: 이항 연산자 (2/3)

예제 0-11: 연산자 다중 정의 -- 이항 연산자

(1/2)

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

class Point
{
public:
    Point(int x = 0, int y = 0);
    void showPosition(void);
    friend Point operator+(int num, Point& p);
};

Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}

void Point::showPosition(void) {
    cout << "(" << x << ", " << y << ")" << endl;
}
```

Point
int x; int y;
Point(int = 0, int = 0); void showPosition(void); Point operator+(int); friend Point operator+(int, const Point &);

```
// 연산자 다중 정의(이항 연산자): p + 10
// p.operator+(int)
// operator+(const Point& p, int)
Point Point::operator+(int num) {
    Point temp(x + num, y + num);
    return temp;
}

// 연산자 다중 정의(이항 연산자): p + 10 == 10 + p
// operator+(int, const Point& p)
Point operator+(int num, Point& p) {
    return p + num;
}
```

// 메서드 함수
// 멤버 함수

연산자 다중 정의: 이항 연산자 (3/3)

예제 0-11: 연산자 다중 정의 -- 이항 연산자

(2/2)

```
int main(void)
{
    Point p1(10, 20);
    cout << "p1(10, 20) ->> p1: ";
    p1.showPosition();

    // p2 = p1 + 10;
    Point p2 = p1 + 10; // Point p2 = p1.operator+(10);
    cout << "p2 = p1 + 10 ->> p2: ";
    p2.showPosition();

    // p2 = 10 + p1;
    p2 = 10 + p1;
    cout << "p2 = 10 + p1 ->> p2: ";
    p2.showPosition();

    return 0;
}
```

Point
int x; int y;
Point(int = 0, int = 0); void showPosition(void); Point operator+(int); friend Point operator+(int, const Point &);

Microsoft Visual Studio 디버그 콘솔

```
p1(10, 20) ->> p1: (10, 20)
p2 = p1 + 10 ->> p2: (20, 30)
p2 = 10 + p1 ->> p2: (20, 30)
```

C:\Users\Clickseo\OneDrive\문서\cpp\Clickseo\64\ 이 창을 닫으려면 아무 키나 누르세요...

연산자 다중 정의: 단항 연산자 (1/3)

● 연산자 다중 정의: 단항 연산자

○ 단항 연산자 다중 정의

- 단항 연산자(Unary Operator): 하나의 피연산자에 대한 연산을 수행하는 연산자
 - 대표적인 이항 연산자: ++, --

Point
int x;
int y;
...

++p

p.operator++()

멤버 함수를 이용한 다중 정의

operator++(Point& p)

일반 함수를 이용한 다중 정의

● 전위와 후위 증감 연산자

++p

p.operator++()

p++

p.operator++(int)

--p

p.operator--()

p--

p.operator--(int)

연산자 다중 정의: 단항 연산자 (2/3)

예제 0-12: 연산자 다중 정의 -- 이항 연산자

(1/2)

```
#include <iostream>
using namespace std;
class Point
{
    int x, y;
public:
    Point(int x = 0, int y = 0);
    void showPosition(void);
```

```
// 멤버 함수에 의한 단항 연산자 다중 정의
Point& operator++() { // ++a
    ++x;
    ++y;
    return *this;
}
Point& operator++(int) { // a++
    ++x;
    ++y;
    return *this;
}
```

```
};
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}
void Point::showPosition(void) {
    cout << "(" << x << ", " << y << ")" << endl;
}
```

```
/* 일반 함수에 의한 단항 연산자 다중 정의
friend Point &operator++(Point &);
Point& operator++(Point& p) {
    ++p.x;
    ++p.y;
    return p;
}
```

// 클래스 내 멤버 함수 정의
// ++a

Point
int x; int y;
Point(int = 0, int = 0); void showPosition(void); Point& operator++() Point& operator++(int)

연산자 다중 정의: 단항 연산자 (3/3)

예제 0-12: 연산자 다중 정의 -- 이항 연산자

(2/2)

```
int main(void)
{
    Point a(10, 20);
    cout << "a: ";
    a.showPosition();

    ++a;    // a.operator++();
    cout << "++a ->> a: ";
    a.showPosition();

    a++;
    cout << "a++ ->> a: ";
    a.showPosition();

    return 0;
}
```

Point
int x; int y;
Point(int = 0, int = 0); void showPosition(void); Point& operator++() Point& operator++(int)

Microsoft Visual Studio 디버그 콘솔

```
a: (10, 20)
++a ->> a: (11, 21)
a++ ->> a: (12, 22)
```

C:\Users\Clickseo\OneDrive\문서\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

연산자 다중 정의: cin, cout

예제 0-13: 콘솔 입출력과 연산자 다중 정의 -- cin, cout

```
#include <iostream>
#include <cstdio>

namespace mystd {
    const char* endl = "\n";
    class ostream {
    public:
        ostream& operator<<(const char* str) {
            printf("%s ", str);
            return *this;
        }
        ostream& operator<<(int num) {
            printf("%d ", num);
            return *this;
        }
        ostream& operator<<(double d) {
            printf("%lf ", d);
            return *this;
        }
    };
    ostream cout;
}

int main(void)
{
    mystd::cout << "Hi~ Clickseo" << mystd::endl;
    mystd::cout << 100 << mystd::endl;
    mystd::cout << 3.14159 << mystd::endl;
    mystd::cout << "Hi~ Clickseo" << 100 << 3.14159 << mystd::endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

Hi~ Clickseo

100

3.141590

Hi~ Clickseo 100 3.141590

C:\Users\Clickseo\OneDrive\문서\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

객체지향 프로그래밍

상속과 다형성



상속의 이해 (1/4)

- **상속(Inheritance)**

- 클래스에 구현된 모든 특성(멤버 변수와 멤버 함수)을 그대로 계승 받아 새로운 클래스를 만드는 기능

- 부모 클래스: 기반 클래스(Base Class), Super Class
- 자식 클래스: 파생 클래스(Derived class), Sub Class

```
class DerivedClass : public BaseClass {  
    // 자동으로 기반 클래스의 멤버 변수와 함수를 소유  
    // 새로운 멤버 변수와 함수의 추가 가능  
};
```

- **protected 멤버**: 상속 클래스에서만 접근을 허용하는 멤버

- 외부 접근: private
- 내부 접근 및 상속 관계: public

상속의 이해 (2/4)

예제 0-14: 상속 클래스의 멤버 접근 -- protected

```
#include <iostream>

class BaseClass {
private:
    int    a;
protected:
    int    b;
};

class DerivedClass : public BaseClass {
public:
    void    setData(void) {
        // error C2248: 'BaseClass::a':
        // private 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
        a = 10;
        b = 20;        // 접근 가능
    }
};

int main(void)
{
    BaseClass    base;

    // error C2248: 'BaseClass::a':
    // private 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
    base.a = 10;

    // error C2248: 'BaseClass::b':
    // protected 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
    base.b = 20;

    return 0;
}
```

상속의 이해 (3/4)

● 3가지 형태의 상속

- BaseClass의 멤버는 DerivedClass로 상속되는 과정에서 접근 권한이 변경된다.

상속 형태 Base Class \	public 상속	protected 상속	private 상속
private	접근 불가	접근 불가	접근 불가
protected	protected	protected	private
public	public	protected	private

(
class DerivedClass : public BaseClass
class DerivedClass : protected BaseClass
class DerivedClass : private BaseClass

상속의 이해 (4/4)

예제 0-15: 상속 클래스의 멤버 접근 -- public, protected, private 상속

```
#include <iostream>

class BaseClass {
private:
    int    a;
protected:
    int    b;
public:
    int    c;
};

class DerivedClass : public BaseClass { // public 상속
public:
    void    setData(void) {
        // error C2248: 'BaseClass::a':
        // private 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
        a = 10;
        b = 20; // 접근 가능
        c = 30;
    }
};

int main(void)
{
    DerivedClass    derived;
    // error C2248: 'BaseClass::a':
    // private 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
    derived.a = 10;
    // error C2248: 'BaseClass::b':
    // protected 멤버('BaseClass' 클래스에서 선언)에 액세스할 수 없습니다.
    derived.b = 20;
    derived.c = 30; // 접근 가능
    return 0;
}
```

상속 클래스: 객체 생성 및 소멸 (1/2)

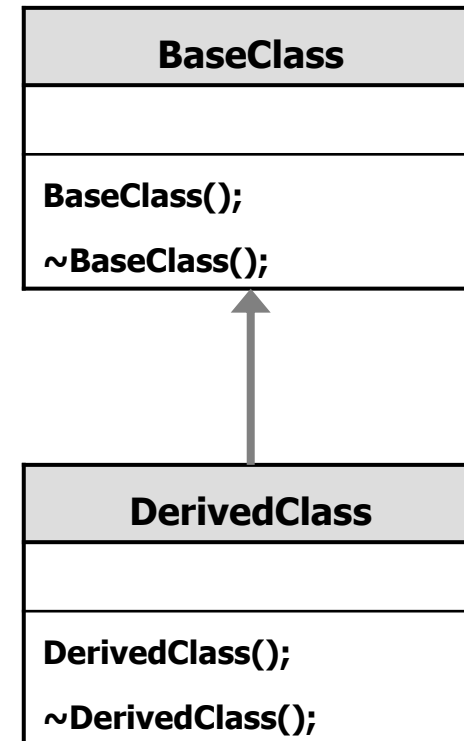
- 상속 클래스: 객체 생성 및 소멸

- 상속 클래스의 객체 생성 과정

1. 메모리 공간의 할당
2. BaseClass 생성자 실행
3. DerivedClass 생성자 실행

- 상속 클래스의 객체 소멸 과정

1. DerivedClass 소멸자 실행
2. BaseClass 소멸자 실행



상속 클래스: 객체 생성 및 소멸 (2/2)

예제 0-16: 상속 클래스의 객체 생성 및 소멸 -- 생성자와 소멸자

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

class BaseClass {
public:
    BaseClass ()    { cout << "Base Class!!!" << endl; }
    ~BaseClass ()   { cout << "~Base Class!!!" << endl; }
};

class DerivedClass : public BaseClass {
public:
    DerivedClass ()      { cout << "DerivedClass Class!!!" << endl; }
    ~DerivedClass ()     { cout << "~DerivedClass Class!!!" << endl; }
};

int main(void)
{
    DerivedClass    derived;
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
Base Class!!!
DerivedClass Class!!!
~DerivedClass Class!!!
~Base Class!!!
```

C:\Users\click\OneDrive\문서\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

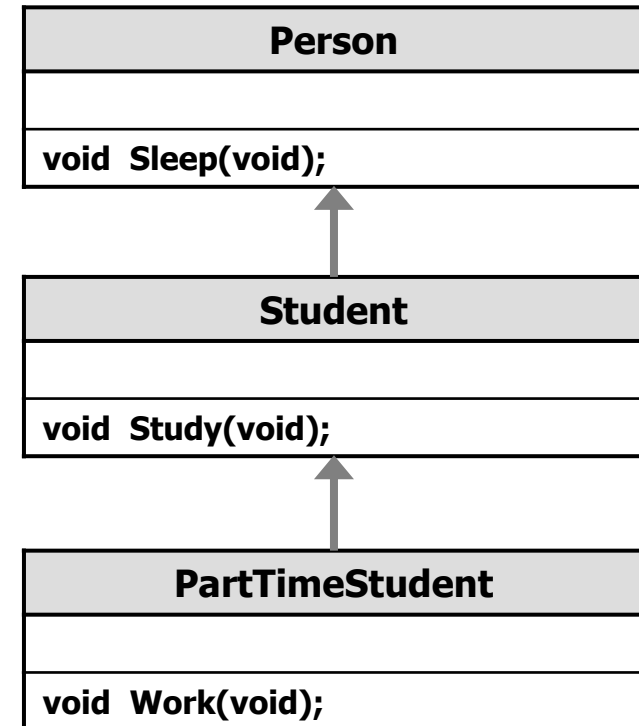
객체의 포인터, 참조 (1/4)

● 객체 포인터와 객체 참조

- **객체 포인터**: 객체의 주소 값을 저장
- **객체 참조**: 객체를 참조할 수 있는 레퍼런스

```
Person* pPerson = new Person;  
Person* pStudent = new Student;  
Person* pPartTimeStudent = new PartTimeStudent;
```

```
PartTimeStudent aPTS;  
Person& rPersonB = aPTS;  
Student& rStudentB = aPTS;  
PartTimeStudent& rPartTimeStudentB = aPTS;
```



“Person 클래스의 객체 포인터(Person *)와 객체 참조 (Person &)는 어떤 대상을 가리키든지,

Person 클래스 내에 선언된 멤버와
Person 클래스가 상속한 클래스의 멤버에만 접근이 가능하다.”

객체의 포인터, 참조 (2/4)

예제 0-17: 객체 포인터와 참조

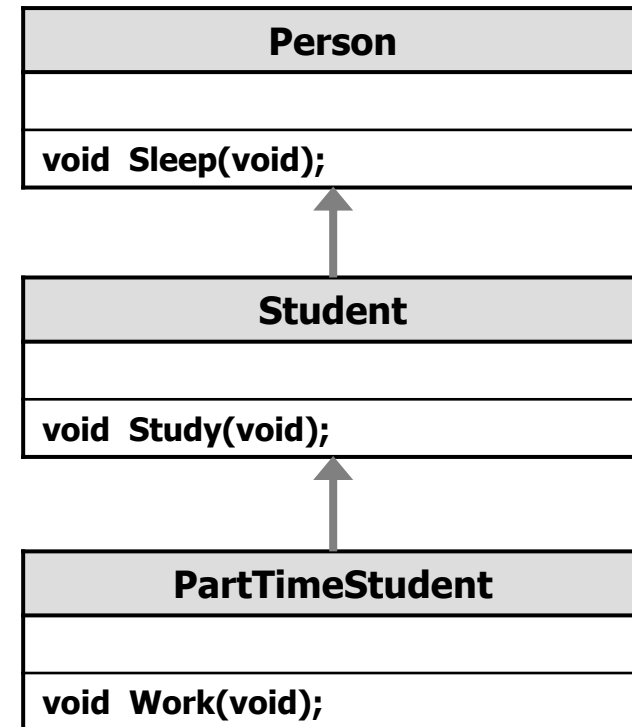
(1/3)

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

class Person {
public:
    void    sleep(void)    {
        cout << "Sleep!!!" << endl;
    }
};

class Student : public Person {
public:
    void    study(void)    {
        cout << "Study!!!" << endl;
    }
};

class PartTimeStudent : public Student {
public:
    void    work(void)    {
        cout << "Work!!!" << endl;
    }
};
```



객체의 포인터, 참조 (3/4)

예제 0-17: 객체 포인터와 참조

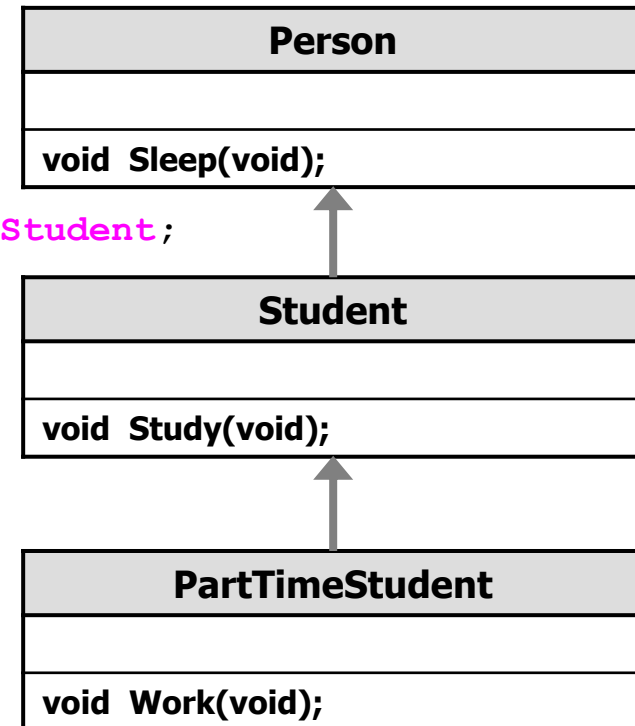
(2/3)

```
int main(void)
{
    // 객체 포인터
    Person* pPerson = new Person;
    Person* pStudent = new Student;
    Person* pPartTimeStudent = new PartTimeStudent;

    pPerson->sleep();
    pStudent->sleep();
    pPartTimeStudent->sleep();

    // error C2039: 'study':
    // 'Person'의 멤버가 아닙니다.
    pStudent->study();

    // error C2039: 'work':
    // 'Person'의 멤버가 아닙니다.
    pPartTimeStudent->work();
}
```



객체의 포인터, 참조 (4/4)

예제 0-17: 객체 포인터와 참조

(3/3)

// 객체 참조

```
PartTimeStudent aPTS;  
Person& rPersonB = aPTS;  
Student& rStudentB = aPTS;  
PartTimeStudent& rPartTimeStudentB = aPTS;
```

```
Person bPTS;  
Person& rPersonA = bPTS;
```

// error C2440: '초기화 중':
// 'Person'에서 'Student &'(으)로 변환할 수 없습니다.

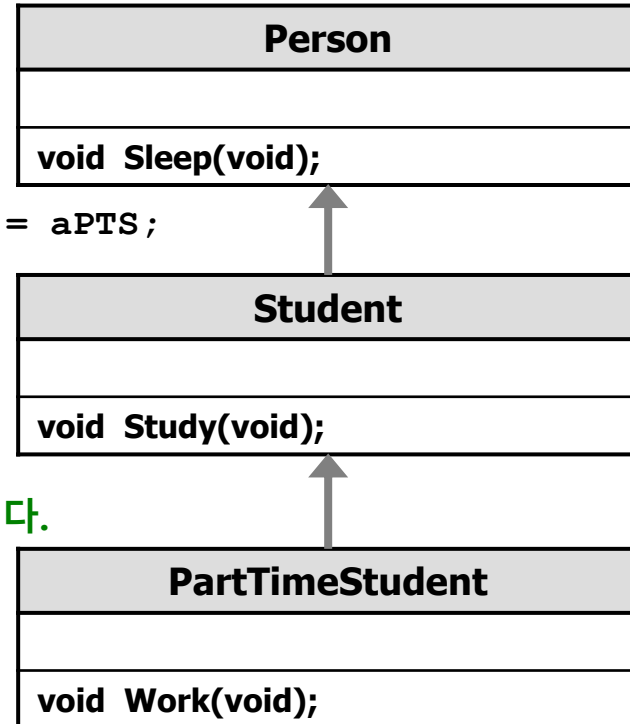
```
Student& rStudentA = bPTS;
```

// error C2440 : '초기화 중':
// 'Person'에서 'Student &'(으)로 변환할 수 없습니다.

```
PartTimeStudent& rPartTimeStudentA = bPTS;
```

```
return 0;
```

```
}
```



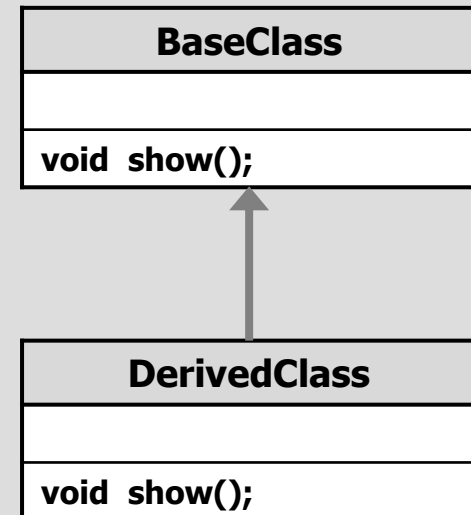
함수 재정의 (1/3)

- **함수 재정의**(Function Overriding)

- 부모 클래스에서 정의된 함수를 자식 클래스에서 다시 정의하여 새로운 기능을 제공한다.

- 이전에 정의된 함수를 숨기는(hide) 특성을 지닌다.

```
class BaseClass {  
public:  
    void show(void) {  
        cout << "BaseClass!!!" << endl;  
    }  
};  
  
class DerivedClass : public BaseClass {  
public:  
    void show(void) {  
        cout << "DerivedClass!!!" << endl;  
    }  
    // 범위 지정 연산자(::)를 통해서 오버라이딩 된 함수도 호출 가능  
    // BaseClass::show()  
};
```



함수 재정의 (2/3)

- 가상 함수(virtual function)

- 함수 재정의된 함수를 가상으로 선언할 수 있다.

```
class BaseClass {  
public:  
    // 가상 함수  
    virtual void show(void) {  
        cout << "BaseClass!!!" << endl;  
    }  
};  
  
class DerivedClass : public BaseClass {  
public:  
    // virtual void show(void)  
    void show(void) {  
        cout << "DerivedClass!!!" << endl;  
    }  
};
```

“가상 함수의 특성은 상속된다.”

함수 재정의 (3/3)

- **순수 가상 함수**(Pure Virtual Function)

- 함수 원형만 가지고 있는 함수
- 추상 클래스(Abstract Class)
 - 하나 이상의 멤버 함수가 순수 가상 함수인 클래스
 - 추상 클래스는 객체를 생성하지 못한다!!!

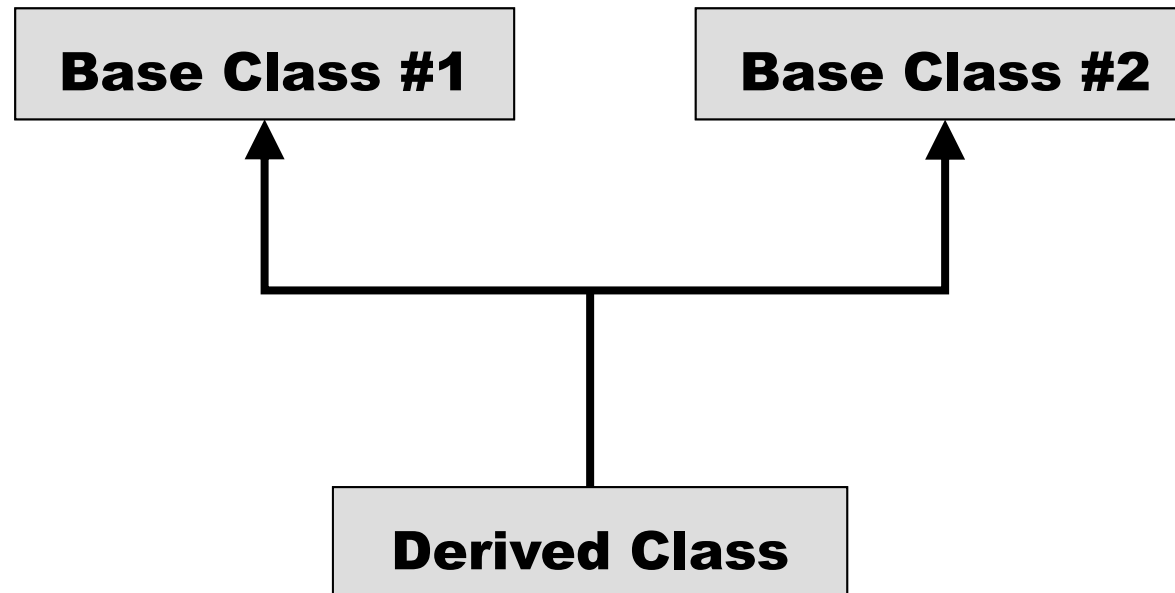
```
// 추상 클래스
class BaseClass {
public:
    // 순수 가상 함수
    virtual void show(void) = 0;    // = 0 : 순수 지정자
};

class DerivedClass : public BaseClass {
public:
    void show(void) {
        cout << "DerivedClass!!!" << endl;
    }
};
```

다중 상속 (1/4)

- **다중 상속**

- 하나의 Derived 클래스가 둘 이상의 Base 클래스를 상속



다중 상속 (2/4)

예제 0-18: 다중 상속

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

```
class SuperClass {
public:
```

```
    void showSuperClass(void) {
        cout << "Super Class!!!" << endl;
    }
};
```

```
class BaseClass {
public:
```

```
    void showBaseClass(void) {
        cout << "Base Class!!!" << endl;
    }
};
```

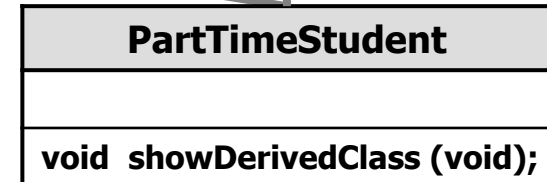
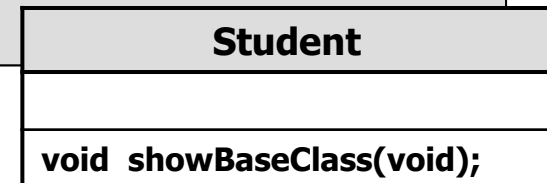
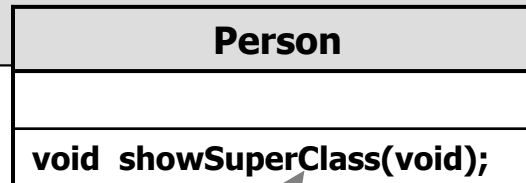
```
class DerivedClass : public SuperClass, public BaseClass {
public:
```

```
    void showDerivedClass(void) {
        cout << "Derived Class!!!" << endl;
        showSuperClass();
        showBaseClass();
    }
};
```

```
int main(void)
{
```

```
    DerivedClass d;
    d.showDerivedClass();
```

```
    return 0;
}
```



Microsoft Visual Studio 디버그 콘솔

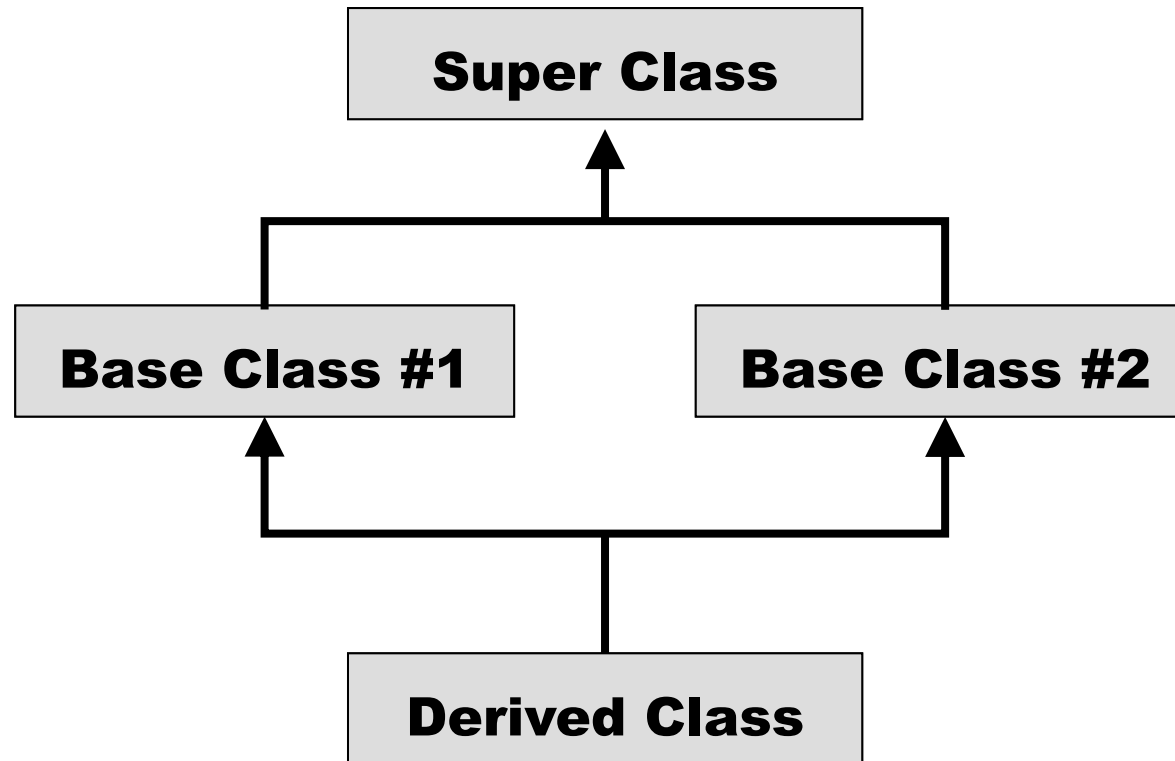
```
Derived Class!!!
Super Class!!!
Base Class!!!
```

다중 상속 (3/4)

- 다중 상속: 모호성

- 다중 상속의 모호성

- Derived 클래스가 간접적인 경로를 통해서 Super Class를 두 번 상속



다중 상속 (4/4)

예제 0-19: 다중 상속 -- virtual 상속

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

class SuperClass {
public:
    void show(void) {
        cout << "Super Class!!!" << endl;
    }
};

class BaseClass01 : virtual public SuperClass {
public:
    void show01(void) {
        cout << "Base Class #1 !!!" << endl;
    }
};

class BaseClass02 : virtual public SuperClass {
public:
    void show02(void) {
        cout << "Base Class #2 !!!" << endl;
    }
};

class DerivedClass : public BaseClass02, public BaseClass01 {
public:
    void showDerivedClass(void) {
        cout << "Derived Class!!!" << endl;
        show();
        show01();
        show02();
    }
};

int main(void)
{
    DerivedClass d;
    d.showDerivedClass();

    return 0;
}
```

객체지향 프로그래밍

예외 처리



예외 처리 (1/8)

● 예외 처리

○ try ... catch

- **try** : 예외 발생에 대한 검사 범위를 설정할 때 사용
- **catch** : 예외를 처리하는 코드 블록을 선언할 때 사용

○ throw

- 예외 상황이 발생하였음을 알릴 때 사용

```
try {  
    if(예외 상황 발생)  
        throw ex;  
}  
catch(exception ex) {  
  
}
```

예외 처리 (2/8)

예제 0-20: 예외 처리 -- 0으로 정수 나누기

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

int main(void)
{
    int    a, b;

    cout << "두 개의 정수 입력: ";
    cin >> a >> b;

    try {
        if(b == 0)
            throw b;
        cout << "a / b = " << a / b << endl;
        cout << "a % b = " << a % b << endl;
    }
    catch(int exception) {
        cout << "입력 오류: " << exception << endl;
        cout << "다시 실행하세요!!!" << endl;
    }

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

두 개의 정수 입력: 10 3

a / b = 3

a % b = 1

C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...

Microsoft Visual Studio 디버그 콘솔

두 개의 정수 입력: 10 0

입력 오류: 0

다시 실행하세요!!!

C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...

예외 처리 (3/8)

예제 0-21: 예외 처리 -- 스택 풀기(Stack Unwinding)

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

int Divide(int a, int b);

int main(void)
{
    int a, b;

    cout << "두 개의 정수 입력: ";
    cin >> a >> b;

    try {
        cout << "a / b = " << Divide(a, b) << endl;
    }
    catch(int exception) {
        cout << "입력 오류: " << exception << endl;
        cout << "다시 실행하세요!!!" << endl;
    }

    return 0;
}

int Divide(int a, int b) {
    if(b == 0)
        throw b;
    return a / b;
}
```

Microsoft Visual Studio 디버그 콘솔

두 개의 정수 입력: 10 3
a / b = 3

C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...

Microsoft Visual Studio 디버그 콘솔

두 개의 정수 입력: 10 0
입력 오류: 0
다시 실행하세요!!!

C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...

예외 처리 (4/8)

예제 0-22: 예외 처리 -- 하나의 try 블록과 여러 개의 catch 블록

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

int main(void)
{
    int    num;

    cout << "정수 입력: ";
    cin >> num;

    try {
        if(num > 0)    throw 1;
        else           throw 'e';
    }
    catch(int exception) {
        cout << "int형 예외 발생: " << exception << endl;
    }
    catch(char exception) {
        cout << "char형 예외 발생: " << exception << endl;
    }

    return 0;
}
```


예외 처리 (5/8)

예제 0-23: 예외 처리 -- 처리되지 않는 예외 처리(abort 함수 호출)

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

int Divide(int, int);

int main(void)
{
    int a, b;

    cout << "두 개의 정수 입력: ";
    cin >> a >> b;

    try {
        cout << "a / b = " << Divide(a, b) << endl;
    }
    catch(char exception) { // char 형 예외 처리!!!
        cout << "입력 오류: " << exception << endl;
        cout << "다시 실행하세요!!!" << endl;
    }

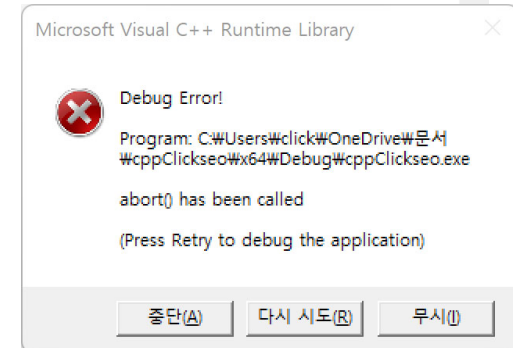
    return 0;
}

int Divide(int a, int b) {
    if(b == 0)
        throw b; // int 형 예외 발생
    return a / b;
}
```

Microsoft Visual Studio 디버그 콘솔

두 개의 정수 입력: 10 0

C:\Users\click\OneDrive\문서\cppClickseo\64\Debug\cppClickseo.exe
(프로세스 17212개)이(가) 종료되었습니다(코드: 3개).
이 창을 닫으려면 아무 키나 누르세요...



예외 처리 (6/8)

- 예외 처리: 전달되는 예외 명시

- 전달되는 예외 명시

```
// throw(type) : until C++ 14
// 함수는 형식 type의 예외를 throw할 수 있습니다.
int fuction(double d) throw (int);
int fuction(double d) throw (int, double, char *);
int fuction(double d) throw (); // 함수는 예외를 전달하지 않는다.
// #pragma warning( disable : 4290 )
// warning C4290: 함수가 __declspec(nothrow)가 아님을 나타내려는 경우를 제외하고 C++
// 예외 사양은 무시됩니다.
// MS C++에서 지원하지 않는 예외(예외 지정을 무시한다. 즉, 예외 지정을 구현하지 않음)
```

```
// 함수는 예외를 throw하지 않습니다(예외 전달 시 abort 함수 호출).
// int fuction(double d) throw (); // C++98
int fuction(double d) noexcept; // noexcept 지정자: since C++ 11
```

예외 처리 (7/8)

● 예외 클래스와 객체

○ 예외를 발생 시키기 위해서 클래스를 정의하고 객체를 생성

- 객체를 이용하면 예외 상황이 발생한 원인에 대한 정보를 자세히 담을 수 있다.

```
class User
{
    string id;
    string pass;
public:
    User(string id, string pass) {
        this->id = id;
        this->pass = pass;
    }
    void ShowUser(void) {
        cout << "아이디 : " << this->id << endl;
        cout << "패스워드: " << this->pass << endl;
    }
};
```

```
try {
    if(ID != userID || PASS != userPASS)
        throw User(userID, userPASS);
}
catch(User& exception) {
    exception.ShowUser();
}
```

예외 처리 (8/8)

- 예외 클래스와 상속

- 예외의 형태가 유사한 경우 예외 클래스를 상속 시키기도 한다.

```
try      {           // 예외 발생!!!
}
catch(exceptionA ex)    {           // 1차 비교
}
catch(exceptionB ex)    {           // 2차 비교
}
catch(exceptionC ex)    {           // 3차 비교
}
```

템플릿과 STL



- C++ 프로그래밍 기초
- 객체지향 프로그래밍
- **템플릿과 STL**
 - Generic 프로그래밍
 - 템플릿
 - STL



Generic 프로그래밍

- **Generic 프로그래밍**

- 정보의 타입과 정보를 처리하는 알고리즘을 분리하는 것

- STL의 컨테이너 클래스들과 알고리즘 함수들

- 컨테이너: 정보의 타입
- 알고리즘 함수: 정보를 처리하는 알고리즘
- sort 함수는 일반적인 배열이나 vector, deque 등과 함께 사용할 수 있게 범용적으로 설계

- 특징

- 타입과 알고리즘 간의 불필요한 연관성 제거
- 재사용성 증가
- 확장이 용이

템플릿과 STL

템플릿



템플릿 (1/6)

- **템플릿**(Template)

- 함수 다중 정의에서 발전된 형태

```
// T 라는 이름(type name)에 대해서, 다음에 정의하는 대상을 템플릿으로 선언
template <typename T>      // 템플릿 정의
T      ADD(T a, T b)  {
    return a + b;
}
```

- 함수 다중 정의의 중복 선언 문제

- 함수의 구현부는 동일하지만, 인자만 다른 여러 함수를 중복하여 선언

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

double   ADD(double a, double b) {
    return a + b;
}
```


템플릿 (2/6)

● 함수 템플릿

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

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

함수 템플릿

```
int main(void)
{
    cout << ADD(10, 20) << endl;           // 정수형(int)으로 인식
    cout << ADD(10.5, 20.5) << endl;       // 실수형(double)으로 인식

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
30
31
C:\Users\click\Downloads\cppClickseo\64\
이 창을 닫으려면 아무 키나 누르세요...
```

템플릿 (3/6)

- 함수 템플릿: 서로 다른 자료형

- 서로 다른 자료형의 템플릿

- 템플릿 매개 변수의 모호함 발생

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

template <typename T>
void ShowData(T a, T b) {
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;
}

int main(void)
{
    ShowData(10, 20);

    // error C2672: 'ShowData': 일치하는 오버로드된 함수가 없습니다.
    // error C2782:
    //     'void ShowData(T,T)': 템플릿 매개 변수 'T'이(가) 모호합니다.
    // error C2784:
    //     'void ShowData(T,T)': 'double'에서 'T'에 대한 템플릿 인수를
    //     추론할 수 없습니다.
    ShowData(10, 10.5);

    return 0;
}
```

```
template <typename T1, typename T2>
void ShowData(T1 a, T2 b) {
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;
}
```

템플릿 (4/6)

예제 0-24: 함수 템플릿의 특수화

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

```
template <typename T>
int SizeOf(T data) {
    return sizeof(data);
}
// 함수 템플릿의 특수화 선언
template<>
int sizeOf(const char* data) {
    return (int)strlen(data);
}
```

// 함수 템플릿의 특수화 선언

```
template<> int sizeOf(char *data)
template<> int sizeOf<>(char *data)
```

```
template<> int sizeOf<char *>(char *data)
```

```
int main(void)
{
    int          i = 10;
    double       d = 10.5;
    char         str[20] = "Hi~ Clickseo!!!";
    const char*  pStr = "Hello World!!!";

    cout << sizeOf(i) << endl;
    cout << sizeOf(d) << endl;
    cout << sizeOf(str) << endl;
    cout << sizeOf(pStr) << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

4
8
8
14

템플릿 (5/6)

● 클래스 템플릿

- 클래스 템플릿은 객체 생성 시 명시적으로 결정하고자 하는 자료형을 선언해야 한다.

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

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

클래스 템플릿

```
int main(void)
{
    Data<int> temp(10);
    cout << "temp: " << temp.getData() << endl;

    temp.setData(20);
    cout << "temp: " << temp.getData() << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
temp: 10
temp: 20
```

템플릿 (6/6)

예제 0-25: 클래스 템플릿

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

```
template <typename T>
class Data
{
public:
    T data;

    Data(T num); // { data = num; }
    void setData(T num); // { data = num; }
    T getData(void); // { return data; }
};

template <typename T> Data<T>::Data(T num) { data = num; }
template <typename T> void Data<T>::setData(T num) { data = num; }
template <typename T> T Data<T>::getData() { return data; }
```

```
int main(void)
{
    // T를 int로 간주하고 객체 생성
    Data<int> a(0); cout << "a: " << a.getData() << endl;
    a.setData(10); cout << "a: " << a.getData() << endl;

    // T를 char로 간주하고 객체 생성
    Data<char> b('F'); cout << "b: " << b.getData() << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

a: 0
a: 10
b: F

C:\Users\Click\Downloads\cppClickseo\64\...
이 창을 닫으려면 아무 키나 누르세요...

템플릿과 STL

STL



STL (1/6)

● **STL** (Standard Template Library)

○ 템플릿을 사용해서 만들어진 클래스와 함수들의 모임

- 일반적으로 많이 사용하는 클래스와 함수
 - 링크드 리스트 클래스, 동적 배열 클래스, 정렬 함수, 검색 함수 등

○ STL의 장점

- **표준:** 개발자들 모두 동일한 코드를 사용한다는 것
- **효율적이고 안전:** 전문가들이 제작

STL (2/6)

- **STL 컨테이너(Containers)**

- 다수의 정보를 담는 역할을 하는 클래스

- 자주 사용하는 STL의 컨테이너 클래스

클래스	내 용
vector	동적인 배열(동적으로 원소의 개수를 조절할 수 있는 배열)
list	링크드 리스트
deque	배열과 링크드 리스트의 장점을 모아 놓은 클래스 (배열만큼 원소에 접근하는 시간이 빠른 동시에 맨 앞과 끝에 원소를 추가하고 제거하는 시간은 링크드 리스트만큼 빠르다.)
map	맵은 원소를 가리키는 인덱스까지도 다양한 타입을 사용할 수 있다.

STL (3/6)

예제 0-26: vector 클래스와 list 클래스

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

// using std::vector;
// using std::list;
using namespace std;

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

    for(int i=1; i<=10; i++)
        v.push_back(i);

    vector<int>::iterator p;
    // list<int>::iterator p;
    for(p = v.begin(); p != v.end(); p++)
        cout << *p << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
1
2
3
4
5
6
7
8
9
10
```

C:\Users\click\Downloads\cppClickseo\64\ 이 창을 닫으려면 아무 키나 누르세요...

STL (4/6)

● STL 알고리즘(Algorithms)

○ STL에서 제공하는 함수

- 정렬이나 검색과 같은 알고리즘을 구현해 놓은 함수

○ 자주 사용하는 STL의 알고리즘 함수

함 수	내 용
<code>find</code>	선형 검색 알고리즘(순차 검색)
<code>replace</code>	특정 값을 가진 원소를 찾아서 다른 값으로 교체한다.
<code>reverse</code>	원소들의 순서를 거꾸로 뒤집는다.
<code>sort</code>	오름차순으로 정렬한다.
<code>binary_search</code>	이진 탐색 알고리즘

STL (5/6)

예제 0-27: 배열과 STL 알고리즘(sort)

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

// using std::sort;
using namespace std;

int main(void)
{
    int    arr[] = { 45, 23, 36, 87, 56 };
    int    arrSize = sizeof(arr) / sizeof(*arr);

    // STL 알고리즘: sort
    sort(arr, arr + arrSize);

    cout << "정렬: ";
    for(int* p = arr; p < arr + arrSize; p++)
        cout << *p << " ";
    cout << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

정렬: 23 36 45 56 87

C:\Users\click\Downloads\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

STL (6/6)

예제 0-28: vector 클래스와 STL 알고리즘(sort)

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

// using std::vector;
// using std::sort;
using namespace std;

int main(void)
{
    vector<int> v;
    v.push_back(5);
    v.push_back(8);
    v.push_back(1);
    v.push_back(3);

    sort(v.begin(), v.end());

    cout << "정렬: ";
    vector<int>::iterator p;
    for(p = v.begin(); p != v.end(); p++)
        cout << *p << " ";
    cout << endl;

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

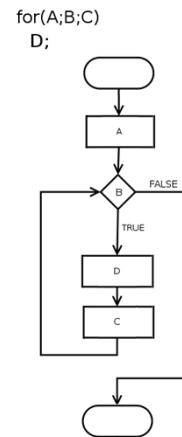
정렬: 1 3 5 8

C:\Users\click\Downloads\cppClickseo\64\이 창을 닫으려면 아무 키나 누르세요...

// STL 알고리즘: sort

참고문헌

- [1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, "C++로 구현하는 자료구조와 알고리즘", 한티에듀, 2020.
- [2] H.M. Deitel, P. J. Deitel, "C++ HOW TO PROGRAM: 10th Edition", Prentice Hall, 2017.
- [3] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.
- [4] 윤성우, "윤성우의 열혈 C++ 프로그래밍"(개정판), 2010.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며, 내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

