

OOP

- in C++ -

안태진(taejin7824@gmail.com)

GitHub(github.com/taejin1221)

상명대학교 소프트웨어학과

201821002

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

Memory Allocation

- Dynamic Allocation




- Compile time이 아닌 Runtime time에 메모리를 할당 받는 방법

- 아니 그래서 메모리 할당이 뭔데?

- 우리가 사용하는 변수는 메모리에 저장
- 하지만 메모리를 모든 프로그램에게 다
- 따라서 현재 실행되는 프로그램에게만
- 장사 할 자리를 내어준다고 생각

- Memory Allocation (메모리 할당)

- 메모리를 내어주는 것 (변수가 저장될 공간을 주는 것, 장사를 하도록 허락)

	1 삼성전자 DDR4-3200			
	데스크탑용 / DDR4 / 램개수: 1개 / 3200MHz (PC4-25600) / 히트싱크: 미포함	32GB 5,524원/1GB	176,770원 <input type="checkbox"/>	266물 <input type="checkbox"/>
	관련기사 10세대 인텔 코어 프로세서로 준비하는, 연말 시즌을 위한 최고의 게이밍 PC	2위 16GB 5,914원/1GB	94,620원 <input type="checkbox"/>	255물 <input type="checkbox"/>
	사용기 삼성전자 DDR4-3200 (8GB)	1위 8GB 5,511원/1GB	44,090원 <input type="checkbox"/>	302물 <input type="checkbox"/>
등록일 2020.06 상품의견 1,212건 관심상품		4GB 7,625원/1GB	30,500원 <input type="checkbox"/>	127물 <input type="checkbox"/>
	2 삼성전자 DDR4-2666			
	데스크탑용 / DDR4 / 램개수: 1개 / 2666MHz (PC4-21300) / 히트싱크: 미포함	32GB 5,743원/1GB	183,780원 <input type="checkbox"/>	224물 <input type="checkbox"/>
	기획전 내 컴퓨터로 8K 재생환경 갖추기	2위 16GB 5,346원/1GB	85,530원 <input type="checkbox"/>	436물 <input type="checkbox"/>
	관련기사 생애 첫 조합하다 게이밍/업무용 컴퓨터 조립하기[브로리퀘스트]	1위 8GB 5,744원/1GB	45,950원 <input type="checkbox"/>	766물 <input type="checkbox"/>
사용기 라이젠 R5 3600 & GTX 1070 기반의 초 가성비 시스템입니다		4GB 7,500원/1GB	30,000원 <input type="checkbox"/>	359물 <input type="checkbox"/>
등록일 2018.03 상품의견 26,869건 관심상품				
	3 GeIL DDR4-2666 CL19 PRISTINE 표준PC			
	데스크탑용 / DDR4 / 램개수: 1개 / 2666MHz (PC4-21300) / 램타이밍: CL19-19-19-43 / 1.20V / 히트싱크: 미포함 / 발크 여부 확인요망	16GB 5,806원/1GB	92,890원 <input type="checkbox"/>	207물 <input type="checkbox"/>
	관련기사 가성비 뛰어난 부품의 조합 '다나와 2월의 게이밍용 표준PC'	1위 8GB 5,873원/1GB	46,980원 <input type="checkbox"/>	268물 <input type="checkbox"/>
	사용기 GeIL DDR4 8G PC4-25600 CL22 PRISTINE 리뷰	2위 4GB 6,750원/1GB	27,000원 <input type="checkbox"/>	223물 <input type="checkbox"/>
등록일 2018.05 상품의견 1,986건 브랜드로그 관심상품				

싸!

Memory Allocation

- 할당되는 Segment
 - Process는 Memory를 다양한 Segment로 분할하여 관리
 - 대표적인 Segment
 - Stack
 - Heap
- Stack Segment
 - 지역 변수들이 저장
- Heap Segment
 - "동적 할당"된 변수들이 저장

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

Stack vs Heap

- Stack Segment의 특징
 - Stack처럼 아래에서 위로 데이터들이 쌓임
- 지역 변수들이 저장되기 때문에 할당될 크기가 예측 가능
 - 프로그램을 실행하기 전에 미리 할당될 크기를 정해놓음
 - 프로그램이 시작할 때 운영체제에게 메모리 요청
 - 바로 할당 받아 실행
 - 빠르다!

Stack vs Heap

- Stack Segment의 단점 (1/4)

1. 메모리의 크기를 꼭 지정해줘야 함

- 가변적으로 메모리를 할당 받고 싶을 때 그러지 못함
- 요즘 Compiler들은 다 해주는데 원래는 안돼요 ㅎ

```
// StackSegment_Prac1.cpp
#include <iostream>

using namespace std;

int main(void) {
    int n;
    cin >> n;

    int arr[n];
    for ( int i = 0; i < n; i++ ) {
        cin >> arr[i];
    }

    return 0;
}
```


Stack vs Heap

- Stack Segment의 단점 (2/4)

2. 메모리의 크기를 알기 때문에 엄청나게 큰 메모리를 할당 받을 때 안해줌

- $4 \times 10,000 \times 10,000 = 400,000,000 \text{ bytes} \approx 400 \text{ MB}$

```
// StackSegment_Prac2.cpp
#include <iostream>

#define MAX_SIZE 10'000

using namespace std;

int main(void) {
    int matrix[MAX_SIZE][MAX_SIZE];

    for ( int i = 0; i < MAX_SIZE; i++ )
        for ( int j = 0; j < MAX_SIZE; j++ )
            matrix[i][j] = i * j;

    cout << "Successfully Done\n";

    return 0;
}
```

```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 g++ -std=c++17 -o StackSegment StackSegment_Prac2.cpp
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./StackSegment
[1] 2376 segmentation fault ./StackSegment
```

Stack vs Heap

- Stack Segment의 단점 (3/4)

- 3. 사용자의 입력은 언제나 다름

- 할당 받았지만 다 쓰지 않으면 낭비가 발생

```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./StackSegment3
1
a
Hello, a!
```

- 할당 받았지만 그것보다 더 원할 수도 있음

```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./StackSegment3
101
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Hello, aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
```

- 딱 필요한 만큼만 할당 받고 싶지만...
 - 1번의 단점 때문에 불가!

```
// StackSegment_Prac3.cpp
#include <iostream>

using namespace std;

int main(void) {
    char string[100];

    int n;
    cin >> n;

    cin >> string;

    cout << "Hello, " << string << "!\n";

    return 0;
}
```

Stack vs Heap

- Stack Segment의 단점 (4/4)
 - 지역을 나가면 변수가 사라짐
 - 지역에서 생성하고 지역 밖에서도 사용하고 싶지만 그럴 수 없음

```
// StackSegment_Prac4.cpp
#include <iostream>

using namespace std;

char* get_string( ) {
    char string[100];
    cin >> string;

    return string;
}

int main(void) {
    char* string = get_string();

    cout << string << '\n';

    return 0;
}
```

```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 g++ -std=c++17 -o StackSegment4 StackSegment_Prac4.cpp
StackSegment_Prac4.cpp:10:9: warning: address of stack memory associated with local variable 'string' returned
    [-Wreturn-stack-address]
    return string;
           ^~~~~~
1 warning generated.

~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./StackSegment4
taejin-tutoring
0500
```

Stack vs Heap

- Heap Segment의 등장 (1/4)
 - 이 전에 설명한 Stack Segment를 모두 커버 가능

1. 메모리의 크기를 변수로 지정 가능

```
// HeapSegment_Prac1.cpp
#include <iostream>

using namespace std;

int main(void) {
    int n;
    cin >> n;

    int* arr = new int[n];
    for ( int i = 0; i < n; i++ )
        cin >> arr[i];

    return 0;
}
```

Stack vs Heap

- Heap Segment의 등장 (2/4)
 2. 메모리가 아무리 커도 할당 받을 수 있음

```
// HeapSegment_Prac2.cpp
#include <iostream>

#define MAX_SIZE 10'000

using namespace std;

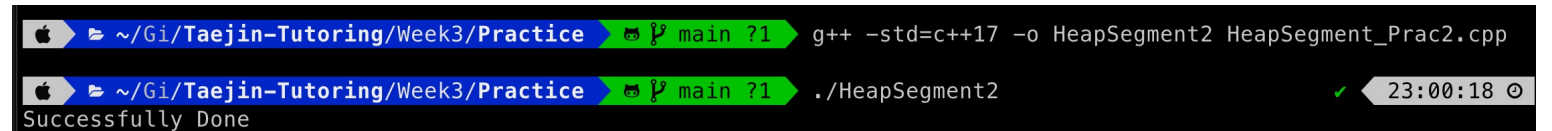
int main(void) {
    int** matrix = new int*[MAX_SIZE];
    for ( int i = 0; i < MAX_SIZE; i++ )
        matrix[i] = new int[MAX_SIZE];

    for ( int i = 0; i < MAX_SIZE; i++ )
        for ( int j = 0; j < MAX_SIZE; j++ )
            matrix[i][j] = i * j;

    for ( int i = 0; i < MAX_SIZE; i++ )
        delete[] matrix[i];
    delete[] matrix;

    cout << "Successfully Done\n";

    return 0;
}
```



```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 g++ -std=c++17 -o HeapSegment2 HeapSegment_Prac2.cpp
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./HeapSegment2 23:00:18
Successfully Done
```

Stack vs Heap

- Heap Segment의 등장 (3/4)
 - 3. 입력 값에 딱 맞는 메모리만 할당 가능

```
// HeapSegment_Prac3.cpp
#include <iostream>

using namespace std;

int main(void) {
    int n;
    cin >> n;

    char* string = new char[n + 1];
    cin >> string;

    cout << "Hello, " << string << "!\n";

    delete[] string;

    return 0;
}
```

[illegible]

Stack vs Heap

- Heap Segment의 등장 (4/4)

4. 지역에서 생성하고 지역 밖에서 얼마든지 사용 가능

```
// HeapSegment_Prac4.cpp
#include <iostream>

using namespace std;

char* get_string( ) {
    char* string = new char[100];
    cin >> string;

    return string;
}

int main(void) {
    char* string = get_string();

    cout << string << '\n';

    delete[] string;

    return 0;
}
```

```
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 g++ -std=c++17 -o HeapSegment4 HeapSegment_Prac4.cpp
~/Gi/Taejin-Tutoring/Week3/Practice main ?1 ./HeapSegment4 ✓ 23:08:03
taejin-tutoring
taejin-tutoring
```

Stack vs Heap

- Static vs Dynamic
 - static allocation
 - 정적 할당, stack 영역에 할당 받는 것
 - dynamic allocation
 - 동적 할당, heap 영역에 할당 받는 것

분야	Static	Dynamic
할당 위치	Stack	Heap
크기 결정 시기	Compile time	Runtime
할당 방향	High -> Low	Low -> High
사용자가 직접 관리?	X	O

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

Dynamic Allocation

- Dynamic Allocation
 - Heap segment에 메모리를 할당 받는 일을 말함
 - Runtime 때 메모리가 할당
 - 사용자가 원할 때 할당 받을 수 있음
 - 높은 자유도
 - 자유에는 책임이 따른다!
 - 할당 받았으면 꼭! 꼭! 메모리를 운영체제에게 돌려줘야함
 - 메모리 해제 (Memory Free)

Dynamic Allocation

- Dynamic Allocation의 단점
 - 사용하기가 복잡
 - 다들 어려워하더라고...
 - 메모리 해제를 까먹어 메모리 누수가 생기기 쉬움
 - 제발 free 해줘 친구들...
 - 코드가 쓸데 없이 길어짐
 - 근데 이거 잘하면 굉장히 있어보임 ㅋㅋㅋㅋ

Dynamic Allocation

- Practice
 - C언어 (외장 함수)
 - 할당
 - stdlib header file의 malloc()
 - 해제
 - stdlib header file의 free()
 - C++ (keyword)
 - 할당
 - new
 - 해제
 - delete

Dynamic Allocation

- Practice 1

- Dynamic Allocation

- Type* variable_name = new Type;

```
char* char1 = new char;  
short* short1 = new short;  
int* int1 = new int;  
long long* long1 = new long long;
```

- Size of Variables

- 단순 Pointer는 주소를 담고 있기 때문

```
cout << sizeof( char1 ) << endl;  
cout << sizeof( short1 ) << endl;  
cout << sizeof( int1 ) << endl;  
cout << sizeof( long1 ) << endl;
```

```
8  
8  
8  
8  
=====
```

- Size of Allocated Memory

- 각 type의 크기 할당

```
cout << sizeof( *char1 ) << endl;  
cout << sizeof( *short1 ) << endl;  
cout << sizeof( *int1 ) << endl;  
cout << sizeof( *long1 ) << endl;
```

```
1  
2  
4  
8  
=====
```

Dynamic Allocation

- Practice 1

- Pointer 사용과 같음
- 할당된 메모리는 꼭! 삭제
 - delete keyword

```
delete char1;  
delete short1;  
delete int1;  
delete long1;
```

```
*int1 = 10;  
cout << int1 << endl;  
cout << *int1 << endl;  
  
cout << "===== " << endl;  
  
*int1 = 20;  
cout << int1 << endl;  
cout << *int1 << endl;
```

```
0x7fe25b405920  
10  
=====  
0x7fe25b405920  
20
```

Dynamic Allocation

- Practice 2

- 여러 값 한번에 할당 (Array)

- `Type* arr = new Type[num];`
 - `Type* arr = new Type[num] { e1, e2, ... }`

```
int* nums_heap1 = new int[5] { 1, 2, 3, 4, 5 };  
int* nums_heap2 = new int[5] { 6, 7, 8, 9, 10 };
```

- 여러 값 한번에 해제

- `delete[] arr;`

```
delete[] nums_heap1;  
delete[] nums_heap2;
```

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

실제 세계는 객체로 이루어져 있으며, 발생하는
모든 사건들은 객체간의 상호작용이다.

OOP란?

- OOP (1/2)
 - Object Oriented Programming의 약자
 - 말 그대로 객체 지향 프로그래밍
 - 객체들을 중심으로 객체들 간 서로 상호작용하도록 프로그래밍 하는 기법
 - 프로그램을 객체들의 모임으로 봄
 - 실생활을 simulation 한다고 생각

OOP란?

- OOP (2/2)
 - 주요 용어
 - "객체", "Object"
 - 주요 언어
 - C++, Java, Python, etc.
 - 많은 언어가 OOP를 지원하고 있음

OOP란?

- 왜 쓰는가?

1. 코드 재사용이 용이
2. 유지 보수가 쉬움
 - 모듈화에 용이
3. 대형 프로젝트에 적합
4. 비슷하지만 약간 다른 객체들을 반복적으로 생성해야 할 때
5. 구조체와 함수를 묶을 수 있다니?!
 - 나의 의견 ㅋㅋㅋㅋ

OOP란?

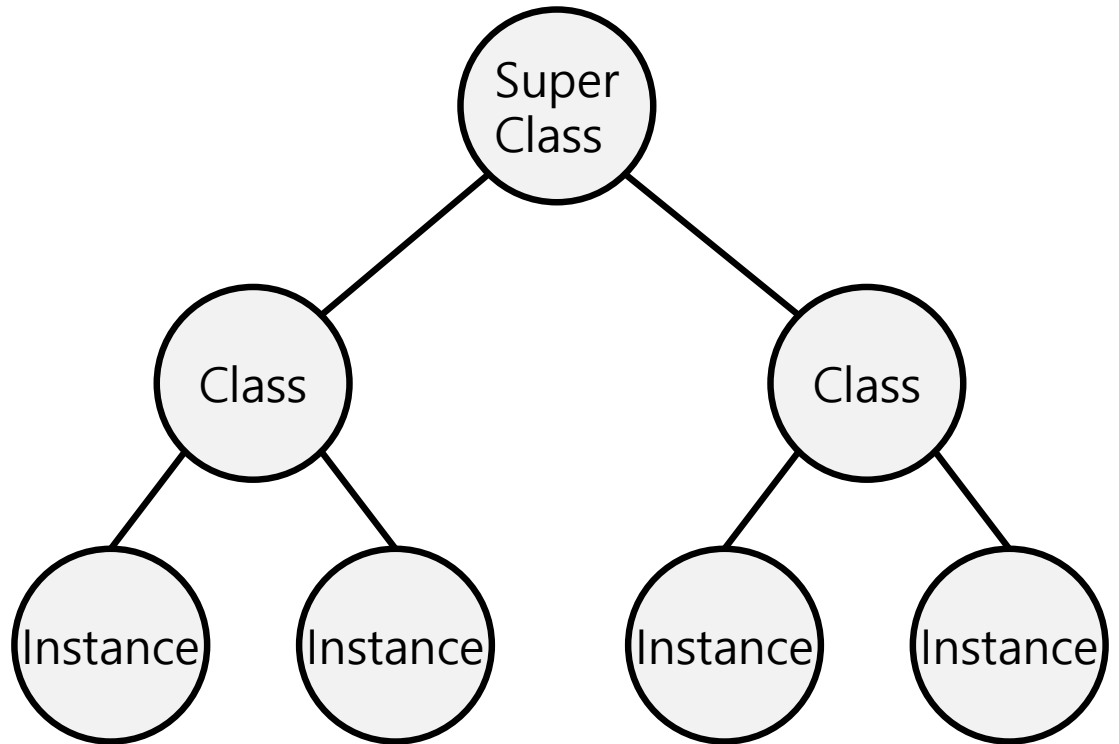
- OOP 구조 (1/2)

- Class

- 객체의 형태를 정해주는 틀
 - 객체를 정의해 놓은 것
 - 객체의 설계도
 - 구조체 + 함수
 - Class에 따라 객체가 만들어짐

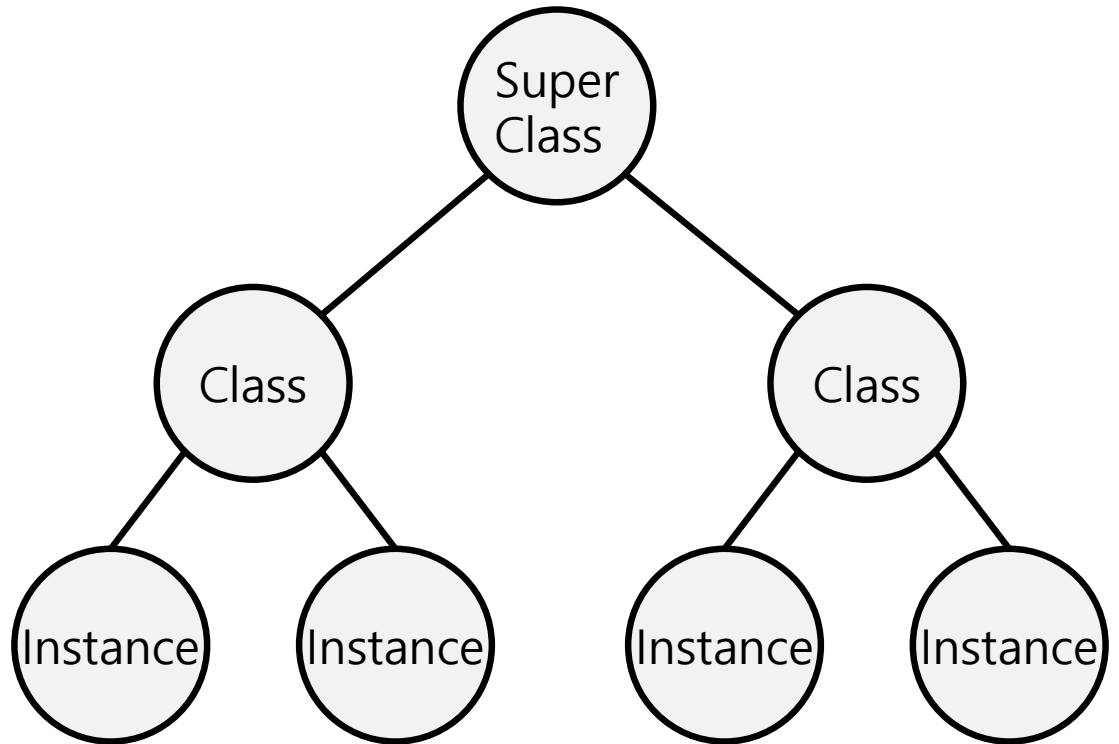
- Object

- Class에 의해 만들어진 객체
 - Instance
 - 실제, 진짜로 만들어진 객체



OOP란?

- OOP 구조 (2/2)
 - Method
 - 객체의 기능
 - 절차 지향 프로그래밍에서의 함수
 - Super class
 - 부모 Class
 - Child class
 - 자식 class



OOP란?

- Class의 구성 요소 (1/2)
 - Member Variable
 - Class가 가지고 있는 변수들
 - Instance들의 상태를 표현
 - Constructor (생성자)
 - 객체가 생성되면 가장 먼저 실행되는 함수
 - 보통 객체의 상태들을 초기화 해주는 역할

OOP란?

- Class의 구성 요소 (2/2)
 - Destructor (소멸자)
 - 객체가 사라질 때 실행되는 함수
 - 보통 메모리 해제를 해주는 역할
 - Method
 - Instance가 실행하는 함수

OOP란?

- Inheritance
 - 자신의 variable 들과 method들을 child class에게 그대로 전달
 - 코드의 재사용성 증가
 - 나중에 설명

OOP란?

- Encapsulation

- 실제 class를 사용하는 사람에게 보여줄 정보, 보여주지 않을 정보를 나누는 것
- 함부로 수정되면 안되는 정보들을 보호하여 프로그래머의 실수 방지
- 보통 private, protected, public 등 존재
 - 오른쪽으로 갈 수록 더 자유로움
 - 언어마다 종류는 다양

OOP란?

- 이론만 알면 소용 없다! 이해 안돼도 괜찮다!
- 이해는 나중에! 구현을 먼저!
- 흐름과 용도를 외우고 이해는 나중에

Contents

- Memory Allocation
- Stack vs Heap
- Dynamic Allocation
- OOP란?
- OOP 실습

OOP 실습

- Phone class 만들 예정
 - field
 - 이름
 - 번호
 - 시리얼 번호
 - 비밀번호
 - 전화번호부
 - 용량
 - method
 - 전화 걸기
 - 전화번호부 보기
 - 전화번호부 추가하기
 - 비밀번호 변경
 - 초기화

OOP 실습

- Class 선언 (1/2)

- class className

- class variable 생성

- class 내부에 type (variableName)

- method 선언

- class 내부에 returnType methodName(type param1, type param2, ...)

```
class Phone {
private:
    string name;
    string phoneNumber;
    string serialNumber;
    string password;
    string* phoneBook;
    int bookSize, bookIdx;

    string encoding( string pw );
    bool isCorrectUser( );
public:

    Phone( string name, string serialNumber, int bookSize );
    Phone( string name, string phoneNumber, string serialNumber, int bookSize );

    ~Phone( );

    string getName( );
    bool hasNumber();
    void callTo( Phone* to );
    void printPhonebook( );
    void addPhoneNumber( string number );
    void changePassword( );
    void init();
};
```

OOP 실습

- Class 선언 (2/2)
 - Constructor
 - class 내부에 `className(type param1, type param2, ...)`
 - Method Overloading
 - Parameter나 return type이 다르면 같은 이름의 method 여러개 생성 가능
 - Destructor
 - class 내부에 `~className(type param1, type param2, ...)`
- 예제
 - Phone.h

OOP 실습

- Class 구현 (1/2)
 1. class 내부 method 파트에 구현
 - 함수 구현하듯이
 2. class 외부에 구현
 - method 이름 앞에 className:: 붙이고 구현
- 예제
 - Phone.cpp

OOP 실습

- Instance 생성
 1. `className variableName(arg1, arg2, ...)`
 - 지역 변수로 선언
 2. `className* variableName = new className(arg1, arg2, ...)`
 - 동적할당으로 선언
- 둘 다 argument에 맞는 constructor 호출

OOP 실습

- Method 사용
 - C언어에서의 구조체와 동일
- 지역 변수
 - `variableName.methodName(arg1, arg2, ...)`
- 동적 변수
 - `variableName->methodName(arg1, arg2, ...)`

OOP 실습

- Instance의 삭제
 - 지역 변수
 - 자동 삭제
 - 동적 변수
 - 꼭 삭제해 주기
 - delete variableName으로 삭제
- 예제
 - OOP_main1.cpp

Reference

- 태진's Brain
- cpluscplus.com
- wikipedia
- [정아마추어 코딩블로그](#)

감사합니다!
