

# C++ Programming

## 6<sup>th</sup> Study: Object-Oriented Programming (2/8)

- Constructor
- Destructor
- this pointer

C++ Korea 옥찬호 (utilForever@gmail.com)



# Constructor

A special type of subroutine called to create an object

# Constructor

- 클래스라면 지원해야 할 기본적인 연산
  - 클래스 자신을 초기화하기 → 생성자
  - 메모리 등 리소스를 정리(청소)하기 → 소멸자
  - 클래스 자신을 복사하기 → 복사 생성자, 복사 할당 연산자
- 객체가 선언될 때 실행되는 코드 : 생성자(Constructor)!
  - 클래스와 같은 이름의 함수, 인수나 리턴 값이 없는 상태로 선언
  - 예를 들어, 클래스의 이름이 Person라면, 생성자는 `Person() { ... }`
  - 생성자는 `public`! → `private`에 있으면, 외부에서 생성자 호출 X (하지만 싱글턴 패턴과 같이 `private`에 선언하는 경우도 존재)

# Constructor

- 생성자가 호출되는 시기
  - 객체를 만들 때 : `Person p;`
  - 메모리를 할당할 때 : `Person* p = new Person;`
- 생성자들의 오버로드도 가능
  - 생성자들의 오버로드 조건은 함수 오버로드의 조건과 동일함
  - 예를 들어, 키와 몸무게를 인수로 가지는 두 번째 생성자를 생성  
`Person(double _height, double _weight);`
  - 이제 인수를 생성자에 넘겨줄 수 있음  
`Person p(174.3, 78.5);`

# Constructor

- 생성자를 만들지 않으면 어떻게 될까?
  - 생성자를 만들지 않더라도 C++은 생성자를 대신 만들어줌  
→ 디폴트 생성자(Default Constructor)
  - 디폴트 생성자는 인수를 가지지 않지만, 멤버가 초기화됨  
(단, int나 char 등의 기본 타입은 초기화되지 않으니 주의)
- 하지만 생성자를 하나라도 만들면, 디폴트 생성자는 만들어지지 않음
  - 컴파일러는 개발자가 어떻게 해야 하는지 충분히 알고 있다고 판단하고, 컴파일러 자신의 도움이 필요 없다고 가정함
  - 예를 들어, `Person(double _height, double _weight);`라는 생성자를 만들었다면, `Person();` 생성자는 자동으로 만들어지지 않으므로 직접 만들어야 함

# Constructor

- 클래스의 멤버 초기화하기
  - 클래스의 각 멤버는 생성자로 초기화되어야 함
    - `Person::Person(double _height, double _weight)`  
`{ height = _height; weight = _weight; }`
  - 이 코드의 내부 동작은 다음과 같음
    - `double height, weight;`  
`height = _height;`  
`weight = _weight;`
    - `Person` 생성자가 시작하는 바로 그 곳에서 `height, weight`이 선언됨  
(클래스 객체의 경우 생성자가 호출됨)
    - 멤버들이 선언되거나 생성자가 호출되어야 사용될 수 있음

# Constructor

- 클래스의 멤버 초기화하기

- 하지만, 어떤 경우에는 문제가 될 수 있음

- const 멤버 변수의 경우 선언할 때 초기화가 되어야 함 (값 변경 X)
    - 레퍼런스 멤버 변수의 경우 어떤 변수를 참조할 것인지 지정해야 함

- 이 때는, 디폴트 생성자를 실행하지 않고 클래스 멤버의 생성자로 인수를 전달해야 함 → 멤버 이니셜라이저(Member Initializer)!

- 생성자 함수에 콜론(:)을 붙인 뒤, “멤버 변수(초기화할 값)”로 표시, 쉼표(,)로 구분

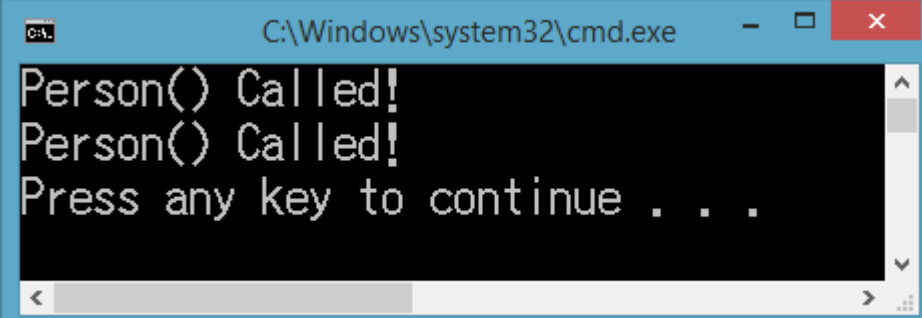
- 예 : `Person::Person(double _height, double_weight)`  
          `: height(_height), weight(_weight) { }`

- 이 코드의 내부 동작은 다음과 같음 : `double height = _height, weight = _weight;`

# Constructor: Example

```
class Person
{
private:
    double height;
    double weight;
public:
    Person() {
        cout << "Person() Called!\n";
    }
};

int main()
{
    Person p1;
    Person* p2 = new Person();
}
```





A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. It displays the output of the C++ program: "Person() Called!" on the first line, "Person() Called!" on the second line, and "Press any key to continue . . ." on the third line. The window includes standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

```
C:\Windows\system32\cmd.exe
Person() Called!
Person() Called!
Press any key to continue . . .
```



# Constructor: Example



```
class Person {  
private:  
    double height;  
    double weight;  
public:  
    Person(double _height, double _weight)  
    {  
        height = _height;  
        weight = _weight;  
    }  
};  
  
int main() {  
    Person p1(183.4, 78.5);  
    Person p2; // Error!  
}
```

	Code	Description ▾
		IntelliSense: no default constructor exists for class "Person"
	C2512	'Person': no appropriate default constructor available

# Constructor: Example

```
class Person {  
private:  
    const string SSN;  
    double height;  
    double weight;  
public:  
    Person() { }  
    Person(const string _SSN, double _height, double _weight) {  
        SSN = _SSN;  
        height = _height;  
        weight = _weight;  
    }  
};  
  
int main() {  
    Person p("123456-1234567", 183.4, 78.5);  
}
```

# Constructor: Example

	Code	Description ▾
		IntelliSense: no operator "=" matches these operands ...
	C2678	binary '=': no operator found which takes a left-hand operand of type 'const std::string' (or there is no acceptable conversion)

# Constructor: Example

```
class Person {  
private:  
    const string SSN;  
    double height;  
    double weight;  
public:  
    Person() { }  
    Person(const string _SSN, double _height, double _weight)  
        : SSN(_SSN), height(_height), weight(_weight)  
    { }  
};  
  
int main() {  
    Person p("123456-1234567", 183.4, 78.5);  
}
```

# Destructor

A method which is automatically invoked when the object is destroyed

# Destructor

- 생성자가 객체를 초기화하듯, 반대로 객체가 더 이상 쓸모가 없을 때 깨끗하게 정리하는 코드가 필요
  - 예를 들어, 생성자가 메모리(또는 다른 리소스)를 할당하면 이 메모리(또는 다른 리소스)는 객체가 더 이상 사용되지 않을 때 운영체제로 리턴되어야 함 → 소멸자(Destructor)!
- 소멸자는 객체가 더 이상 필요 없을 때 호출됨
  - 예를 들어, 객체의 포인터에 delete를 호출할 때
- 소멸자 선언 방법 : 생성자 앞에 틸트(~) 문자를 붙임
  - 예를 들어, 클래스의 이름이 Person라면, 소멸자는 ~Person() { ... }

# Destructor

- 객체가 ‘더 이상 필요하지 않을 때’?
  - 삭제될 때 파괴
    - `Person* p = new Person();`  
`delete p;`
  - 영역 밖으로 넘어갈 때 파괴
    - `if (true) {`  
    `Person p;`  
    `} // p의 소멸자 호출`

# Destructor

- 객체가 ‘더 이상 필요하지 않을 때’?
  - 다른 소멸자에 의한 파괴
    - ```
class Person {  
    private:  
        string name;  
        string ssn;  
};
```
    - name과 ssn의 파괴자는 Person 소멸자가 실행을 마친 뒤 호출
    - 클래스에 소멸자를 추가하지 않더라도 클래스의 객체에 대해 소멸자를 실행해 줌
    - 리소스 할당은 초기화! : RAI(Resource Allocation Is Initialization)  
생성자로 클래스를 초기화하고 소멸자로 클래스 소유의 리소스를 정리한다는 개념



# Destructor

- 생성 순서와 소멸 순서
  - 다음과 같은 예제가 있을 때...

```
int main()
{
    Person a;
    Person b;

    // ...
}
```

- 생성 순서는 a, b (먼저 선언한 객체가 먼저 생성)
- 소멸 순서는 b, a (먼저 선언한 객체가 나중에 소멸)

# Destructor: Example

```
class IntPtrter
{
private:
    int* pi;

public:
    IntPtrter()
    {
        pi = new int[5];
    }

    ~IntPtrter()
    {
        delete[] pi;
        pi = nullptr;
    }
};
```

# Destructor: Example

```
class A {  
public:  
    A() { cout << "A() Called!\n"; }  
    ~A() { cout << "~A() Called!\n"; }  
};  
  
class B {  
public:  
    B() { cout << "B() Called!\n"; }  
    ~B() { cout << "~B() Called!\n"; }  
};  
  
int main() {  
    A a;  
    B b;  
}
```

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The output of the program is displayed as follows:  
A() Called!  
B() Called!  
~B() Called!  
~A() Called!  
Press any key to continue . . .  
The window has a standard Windows title bar with minimize, maximize, and close buttons.

# this pointer

A prvalue expression whose value is the address of the object, on which the member function is being called

# this pointer

- this 포인터는 클래스에서 사용할 수 있는 특별한 포인터!
  - 현재 멤버 함수가 호출된 인스턴스의 메모리 주소를 가리킴
- this 포인터는 멤버 변수들과 멤버 함수들을 연결해줌
  - 멤버 변수들은 각각의 인스턴스에서 저장할 내용이 다르기 때문에 반드시 별도로 존재해야 하지만, 멤버 함수들은 인스턴스가 아무리 늘어나더라도 바뀔 필요가 없음
  - 프로세스 구조상 멤버 변수들이 보관되는 영역(스택, 힙, 데이터)과 멤버 함수들이 존재하는 영역(코드)은 나누어져 있음
  - 코드 영역은 실행 중에 변경을 막기 위해 보호되어 런타임 중 변경 X

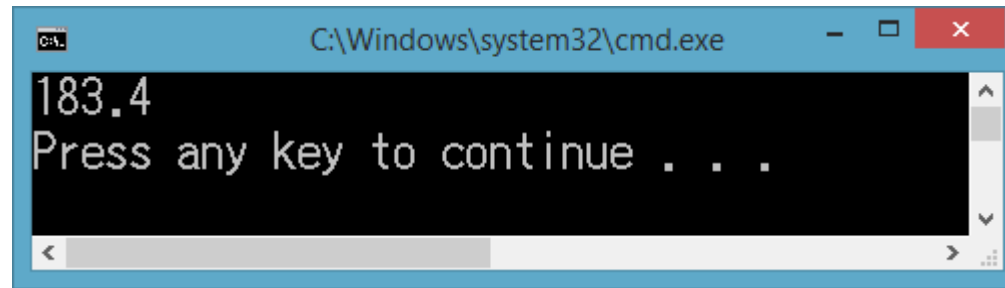
# this pointer

- 인스턴스는 독자적인 멤버 변수들을 가지지만, 클래스 공통의 멤버 함수와 매칭됨
  - 이 때, 멤버 함수가 인스턴스를 구별하는 방법이 this 포인터!
  - 멤버 함수를 호출하게 되면, 멤버 함수에 호출한 인스턴스의 포인터를 같이 보내고, 멤버 함수는 인스턴스의 포인터(this 포인터)를 가지고 멤버 변수들에 접근함
- 참고 : 멤버 함수 내에서 명칭의 우선순위는 “지역 변수 > 멤버 변수 > 전역 변수” 순!
  - 멤버 변수와 지역 변수의 이름이 같다면, 지역 변수 우선 순위가 높음!

# this pointer: Example

```
class Person {  
private:  
    double height;  
    double weight;  
public:  
    Person() { }  
    Person(double _height, double _weight)  
        : height(_height), weight(_weight) { }  
    void setHeight(double height) { height = height; }  
    double getHeight() { return height; }  
};  
  
int main() {  
    Person p(183.4, 78.5);  
    p.setHeight(182.8);  
    cout << p.getHeight() << endl;  
}
```

# this pointer: Example





# this pointer: Example

```
class Person {
private:
    double height;
    double weight;
public:
    Person() { }
    Person(double _height, double _weight)
        : height(_height), weight(_weight) { }
    void setHeight(double height) {this->height = height; }
    double getHeight() { return height; }
};

int main() {
    Person p(183.4, 78.5);
    p.setHeight(182.8);
    cout << p.getHeight() << endl;
}
```

# this pointer: Example

