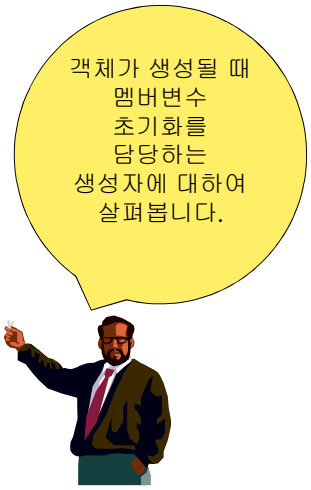
C++ Espresso

제6장 생성자와 소멸자

이번 장에서 학습할 내용

- 생성자
- 소멸자
- 초기화 리스트
- 복사 생성자
- 디폴트 멤버 함수



생성자, constructor

- 생성자(contructor):
  - 객체가 만들어지며 자동 호출되는 멤버 함수
  - 객체가 생성될 때에 멤버변수에게 초기값을 저장하는 역할을 하는 멤버 함수

멤버변수 값을 초기화
- 배기량 ← 2000
- 색 ← red
- 속도 ← 0

객체의 일생

그림 10.2 객체의 일생

멤버변수 초기화 꼭 필요

멤버변수가 포인터인 경우만 필요

생성자의 특징

- 함수 이름이 클래스 이름과 동일하다
- 반환값을 작성하지 않는다. → void 도 아님
- 반드시 public 이어야 한다.
- 중복 정의할 수 있다. → 이름은 같고 매개변수가 다른 생성자 함수들 가능

```
class Car
{
        ...
public:
        Car(...)
        {
                ...
        }
};
```

생성자

디폴트 생성자→ 매개변수 없는 생성자

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

class Car {
private:
        int speed; // 속도
        int gear; // 기어
        string color; // 색상
public:
        Car() {    } // 작성하지 않으면 디폴트 생성자 자동 생성, 하는 일 없음
};
int main() {
        Car c1; // 디폴트 생성자 호출
        return 0;
}
```

- 디폴트 생성자를 작성하지 않는 경우 자동 생성(매개변수 없음, 하는 일 없음)
- 그러나 디폴트 생성자를 작성해야함(다음쪽)

디폴트 생성자→ 매개변수 없는 생성자

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

class Car {
private:
        int speed; // 속도
        int gear; // 기어
        string color; // 색상
public:
        Car() {    // 생성자 작성하면 매개변수 없는 디폴트 생성자 자동 생성 안됨
                cout << "디폴트 생성자 호출" << endl;
                speed = 0;
                gear = 1;
                color = "white";
        }
};
int main() {
        Car c1; // 디폴트 생성자 호출
        return 0;
}
```

speed	0
gear	1
color	"white"

c1

- 생성자는 멤버변수 초기화를 담당하므로 꼭 필요

생성자의 외부 정의

```
class Car {
private:
        int speed; // 속도
        int gear; // 기어
        string color; // 색상
public:
        Car();     // 원형, 반환값 없음.
};

// 다른 멤버함수와 같이 클래스 안에 있는 것을 밖으로 꺼낼 수 있음.

Car::Car() {
        cout << "디폴트 생성자 호출" << endl;
        speed = 0;
        gear = 1;
        color = "white";
}
```

매개 변수를 가지는 생성자(일반적)

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

class Car {
private:  int speed;     // 속도
          int gear;      // 기어
          string color;  // 색상

public:
     Car(int s, int g, string c)
     {
          speed = s;
          gear = g;
          color = c;
     }
     void print()
     {    cout << "============== " << endl;
          cout << "속도: " << speed << endl;
          cout << "기어: " << gear << endl;
          cout << "색상: " << color << endl;
          cout << "============== " << endl;
     }
};
```

```
==============
속도: 0
기어: 1
색상: red
==============
==============
속도: 0
기어: 1
색상: blue
==============
```

```
int main()
{         // 객체이름(멤버변수 초기값들)
     Car c1(0, 1, "red");
     Car c2(0, 1, "blue");
     c1.print();
     c2.print();
     return 0;
}
```

c1(객체)
speed, gear, color
Car(…){
  speed = s;
  gear = g;
  color = c;… }
print( ){ … }

c2(객체)
speed, gear, color
Car(…){ … }
print( ){ … }

생성자의 중복 정의

- 생성자도 다른 함수와 같이 중복 정의가 가능하다.

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

class Car {
private:
     int speed;     // 속도
     int gear;      // 기어
     string color;  // 색상
public:
     Car();
     Car(int s, int g, string c) ;
};
```

```
Car::Car()
{
     cout << "디폴트 생성자"<< endl;
     speed = 0;
     gear = 1;
     color = "white";
}
Car::Car(int s, int g, string c)
{
     cout << "매개변수 있는 생성자" << endl;
     speed = s;
     gear = g;
     color = c;
}
int main()
{
     Car c1;
     Car c2(100, 0, "blue");
     return 0;
}
```

생성자의 중복 정의

- 매개변수 있는 생성자를 작성하면 디폴트 생성자는 자동 생성되지 않음.

→ 항상 매개변수 없는 생성자 Car( ) { … } 를 작성해야 함.

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

class Car {
private:
      int speed;     // 속도
      int gear;      // 기어
      string color; // 색상
public:
      Car();  // 항상 작성 필요
      Car(int s, int g, string c) ;
};
```

```
Car::Car(int s, int g, string c)
{
      cout << "매개변수 있는 생성자" << endl;
      speed = s;
      gear = g;
      color = c;
}
int main()
{
      Car c1;      // Error , 생성자 본체 없음
      Car c2(100, 0, "blue");
      return 0;
}

Car::Car(){ // 항상 작성 필요
            speed = 1;  gear = 1;    color = "red";
} // 매개변수 없이 객체 생성하는 경우 많음
```

생성자 호출(객체 생성)의 다양한 방법

```
// Car 클래스는 이전 쪽 내용
int main(){
      Car c1;   // ①디폴트 생성자 호출
      Car c2();// ②이것은 생성자 호출이 아니라 c2()라는 함수의 원형 선언, error
      Car c3(100, 3, "white");        // ③생성자 호출
      Car c4 = Car(0, 1, "blue");     // ④ 먼저 이름이 없는 임시 객체를 만들어
                                      // 멤버변수 초기화후 c4라는 이름으로 사용( 복사생성자 호출 안함)

      // 포인터 변수(객체)인 경우 new 사용 객체 생성
      Car *c5_0 = new Car;  // 디폴트 생성자 호출
      Car *c5 = new Car( );  // 디폴트 생성자 호출
      Car *c6 = new Car(20, 11, "blue");

      return 0;
}
```

c4, c6 → 동일하게 생각하면 됨 → 둘 다 이름 없는 객체 생성
- c4 라는 이름으로 사용,
- 객체주소를 c6에 저장

```
int sum( );
Car c2( );     // 함수 원형

int Sum( ){
  return (10+20);
}

Car c2( ){  Car x;   ....
 return x;
}

int main( ) {
  Sum();
  c2( );
...
}
```

참고 객체 생성 정리 2 (C++ vs C#)..

- C++
  - Car c1; // int x;
  - Car c2( ); // Error
  - Car c3(100, 3, "white");
  - Car c4 = Car(0, 1, "blue");
  - Car obj[3]; // int y[3];

    // 포인터 변수인 경우 new 사용 객체 생성
  - Car *c51 = new Car; // 디폴트 생성자 호출
  - Car *c5 = new Car( ); // 디폴트 생성자 호출
  - Car *c6 = new Car(20, 11, "blue");
  - Car *c7 = new Car[3]; // 동적 배열

- C#
  - Car c1, c2;
  - Car c3=new Car( );
  - Car[ ] c5 = new Car[10]; 등등

13

생성자를 하나도 정의하지 않으면?

```
class Car {
    int speed;      // 속도
    int gear;       // 기어
    string color;   // 색상
};
```

컴파일러가 디폴트 생성자를
자동으로 추가한다.→ but, 작성해야함

```
class Car {
    int speed;      // 속도
    int gear;       // 기어
    string color;   // 색상
public:
    Car() { }
}
```

디폴트 매개 변수..

```
class Car {
private:
  int speed;            // 속도
  int gear; // 기어
  string color;         // 색상
public:
  // 생성자
  Car(){ }              // ERROR
  Car(int s=0, int g=1, string c="red") ;
};

Car::Car(int s, int g, string c) {
  speed = s;
  gear = g;
  color = c;
}
```

```
int main(){

  Car c2(100, 0, "blue");
  Car c3(10), C4(200, 2);

  Car c1;     // 어느 생성자가 호출 ??

  return 0;
}
```

- 위와 같은 경우는 디폴드 생성자 작성 안함

생성자에서 다른 생성자 호출하기(비중요)

```
...
class Car {
    int speed, gear; // 속도, 기어
    string color; // 색상
public:
    // 첫 번째 생성자
    Car(int s, int g, string c) {
        speed = s;
        gear = g;
        color = c;
    }
    // 색상만 주어진 생성자, → 이 방식은 잘 사용하지 않음.
    Car(string c) {
        Car(0, 0, c);          // 첫 번째 생성자를 호출한다.
    }
};
int main()
{
    Car c1("white");
    return 0;
}
```

소멸자(destructor)

- 객체가 없어질 때 자동 호출되는 멤버함수(함수 벗어나면 변수/객체 소멸)
- 일반적으로 작성 안함 → 구성하지 않아도 자동 생성
- 멤버변수가 포인터인 경우에는 작성해야 함

그림 10.4 소멸자의 개념

생성자
포인터 멤버변수가
가리키는 공간 생성

소멸자
포인터 멤버변수가
가리키는 공간 제거

소멸자, 생성자 정리

- 클래스 내에
- 일반 멤버 변수 존재 + 포인터 멤버 변수 없는 경우
  - 생성자 → 멤버 변수 변수 초기화
  - 소멸자 → 작성 안함(컴파일러가 알아서 소멸자 생성)
- 일반 멤버 변수 존재 + 포인터 멤버 변수 존재
  - 생성자 → 멤버 변수 변수 초기화 + 포인터 멤버변수에 공간 할당
  - 소멸자 → 포인터 멤버변수에 할당된 공간 제거
- 메모리 관리 측면에서 효율적(메모리 오류 가능성 매우 적음)
  - 객체 생성시 메모리 공간 생성
  - 객체 소멸시 메모리 공간 제거

소멸자의 특징

- 소멸자 함수 이름은 클래스 이름에 ~가 붙는다.
- 값을 반환하지 않는다.
- public 멤버 함수로 선언된다.
- 소멸자는 매개 변수를 받지 않는다.
- 중복 정의도 불가능하다.

```
class Car
{
        ...
public:
        ~Car()
        {
                ...
        }
};
```

소멸자

소멸자

```
class Car {
private:
        int speed, gear;      // 속도, 주행거리
        string color;         // 색상
public:
        Car() {
                cout << "생성자 호출" << endl;
                speed = 0;
                gear = 1;
                color = "white";
        }
        ~Car() {
                cout << "소멸자 호출" << endl;
        }
};
int main()
{
        Car c1;
        return 0;
}
```

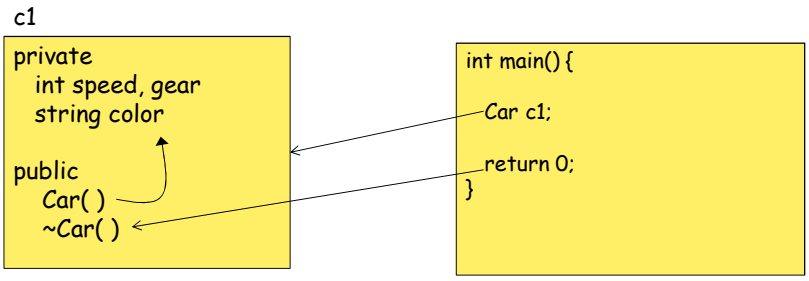
생성자

소멸자

c1
speed=0
gear=1
color="white"

생성자 호출
소멸자 호출

멤버변수/함수 생성자, 소멸자 수행 순서



- c1 객체 생성
  - 멤버변수/함수 생성
  - 생성자 수행 → 변수들 초기값 지정
- 함수 빠져 나오면 c1 객체의 소멸자 수행

디폴트 소멸자

- 만약 프로그래머가 소멸자를 정의하지 않았다면 어떻게 되는가?
- 디폴트 소멸자가 자동으로 삽입되어서 호출된다

```
class Time {
        int hour, minute, second;
public:
        print() { ... }
}
```

~Time( ) { }을 넣어준다.

<정 리>
- 생성자는 작성해야 함(모든 멤버변수 초기화)
- 소멸자는 작성 안해도 됨(멤버변수가 포인터인 경우는 작성해야 함)

참고) C++ 스타일의 초기화

C 스타일 초기화

```
int num=20;
int &ref=num;
```

C++ 스타일 초기화

```
int num(20);
int &ref(num);
```

- 이렇듯, 다음 두 문장은 실제로 동일한 문장으로 해석된다.

  Car y=x; // ... (1)

  Car y(x); // ... (2)

- (1)과 같이 작성하면 내부적으로 (2)로 자동변환 되어짐

```
class Car {
    int speed, gear; // 속도, 기어
    string color; // 색상
    ...
```

멤버 초기화 목록

- 멤버 변수를 간단히 초기화할 수 있는 형식

```
int speed(s);
int gear(g);
string color(c);
```

```
Car(int s, int g, string c) : speed(s), gear(g), color(c) {
        ...// 만약 더 하고 싶은 초기화가 있다면 여기에
}
```

이후에는 생성자에서 멤버변수 초기화는
아래와 같이 하기 보다는 위와 같이 하는 것이 더 좋음..

```
Car(int s, int g, string c)      {
  speed = s;
  gear = g;
  color = c;
}
```

상수 멤버(const)의 초기화

- 멤버가 상수인 경우에는 어떻게 초기화하여야 하는가?

```
class Car
{
        const int MAX_SPEED = 300;  // error
        int speed;                  ...
}
```

아직 생성이 안됐음!
→ 설계도 이기에

```
class Car
{
        const int MAX_SPEED;
        int speed;          // 속도
public:
        Car()
        {
                MAX_SPEED = 300;   // ERROR, C# 가능

        }
}
```

상수를 변경할 수 없음!

상수 멤버의 초기화 방법

```
class Car
{
        const int MAX_SPEED;
        int speed;          // 속도
public:
        Car() : MAX_SPEED(300)
        {
        }
};
```

상수 멤버의 초기화는 이렇게.

const int MAX_SPEED = 300; 으로 해석

상수 멤버는 객체 생성된 후(생성자 호출 후)
멤버초기화 목록을 사용하여 초기화만 가능

참조자 멤버의 초기화

- 포인터, 참조자(3장 참고)

```
int a, *b, &c=a;

a = 20;
b = new int;
*b = 30;

cout << a << b << *b << c;

a = 40;
cout << c;
```

	주소	값
a c	1000	20 → 40
b	1001	1002
	1002	30
	1003	
	1004	

참조자 멤버의 초기화

```
#include <iostream>
#include <string>
using namespace std;
class Car
{
        string& alias;
        int speed;          // 속도
public:
        Car(int sp, string s) : speed(sp), alias(s)  // string &alias=s; 로 해석
        {
                cout << alias << endl;
        }
};

int main()
{
        Car c1(100, "꿈의 자동차");
        return 0;
}
```

클래스 안의 참조자 멤버변수는
- 객체 생성된 후(생성자 호출 후)에야 변수 지정 가능
- 멤버초기화 목록 사용하여야만 초기화 (상수 멤버와 동일)

➔ 무조건 모든 멤버변수는 모두 초기화 목록으로 값 지정

```
꿈의 자동차
계속하려면 아무 키나 누르십시오 . . .
```

예제 1(231쪽) 실습

- Date 클래스
- 멤버변수 : private
  int year;
  int month;
  int day;
- 멤버함수 : public

```
Date();                          // 멤버변수를 2010.1.1
Date(int year);                  // 연도 이외 멤버변수를 1.1
Date(int year, int month, int day);    // 멤버변수를 매개변수로 지정

void setDate(int year, int month, int day); // 멤버변수를 매개변수로 지정
void print();   // 년, 월, 일 을 출력
```

어떤 생성자 필요 ?

```
int main()
{
   Date date1(2009, 3, 2);     // 2009.3.2
   Date date2(2009);           // 2009.1.1
   Date date3;                 // 2010.1.1
   date1.print();
   date2.print();
   date3.print();
   return 0;
}
```

```
2009년 3월 2일
2009년 1월 1일
2010년 1월 1일
```

예제 1

```
class Date {
private:
        int year;
        int month;
        int day;
public:
        Date();                                  // 디폴트 생성자 → 매개변수 없는 것
        Date(int year);                          // 생성자
        Date(int year, int month, int day);      // 생성자
        void setDate(int year, int month, int day);   // 멤버함수
        void print();                            // 멤버함수
};

void Date::setDate(int year, int month, int day) {
        // month = month 는 error
        this->month = month;          // this는 현재 객체(호출한 객체)를 가리킨다.

        this->day = day;
        this->year = year;
}
```

예제 1

```
// 디폴트 생성자
Date::Date() : year(2010), month(1), day(1) {
            // SetData(2010, 1, 1); 해도 무방
}
Date::Date(int y) : year(y), month(1), day(1) {     }

// 매개변수를 멤버변수 이름과 동일하게 작성하지 말 것.
// error 는 아니지만 혼돈
Date::Date(int year, int month, int day) {// 생성자
            setDate(year, month, day);
}
→ 이런 식으로 사용 안함

void Date::print() {
   cout << year << "년" << month <<  "월" << day <<
    "일" << endl;
}
```

```
int main()
{
   Date date1(2009, 3, 2);      // 2009.3.2
   Date date2(2009);            // 2009.1.1
   Date date3;                  // 2010.1.1
   date1.print();
   date2.print();
   date3.print();
   return 0;
}
```

```
2009년 3월 2일
2009년 1월 1일
2010년 1월 1일
```

예제 1 - 더 간단한 풀이

```
class Date {
private:
    int year;
    int month;
    int day;
public:
    // 생성자, 하나로 통일
    Date(int year=2010, int month=1, int day=1);
    void print();
};

Date::Date(int y, int m, int d)
: year(y), month(m), date(d) { }

void Date::print() {
   cout << year << "년" << month <<  "월" << day <<
    "일" << endl;
}
```

```
int main()
{
   Date date1(2009, 3, 2);      // 2009.3.2
   Date date2(2009);            // 2009.1.1
   Date date3;                  // 2010.1.1
   date1.print();
   date2.print();
   date3.print();
   return 0;
}
```

```
2009년 3월 2일
2009년 1월 1일
2010년 1월 1일
```

예제 2

- Time 클래스
- 멤버변수
  int hour; // 0 - 23
  int minute; // 0 - 59
  int second; // 0 - 59

- 멤버함수 :
  Time(); // 생성자, 멤버변수에 0, 0, 0 저장,
  // SetTime 이용
  Time(int h, int m, int s); // 멤버변수에 인자값 저장

  void setTime(int h, int m, int s); // 멤버변수에 인자값 저장
  // 인자 전달 값이 잘못된 시간이면 해당 항목 0으로 저장
  void print();

예제 2

```
int main(){
        Time time1;
        cout << "기본 생성자 호출 후 시간:  ";
        time1.print();

        // 두번째 생성자 호출
        Time time2(13, 27, 6);
        cout << "두번째 생성자 호출 후 시간: ";
        time2.print();

        // 올바르지 않은 시간으로 설정해 본다.
        Time time3(99, 66, 77);          // 시, 분, 초
        cout << "올바르지 않은 시간 설정 후 시간: ";
        time3.print();
        return 0;
}
```

```
기본 생성자 호출 후 시간:  0:0:0
두번째 생성자 호출 후 시간: 13:27:6
올바르지 않은 시간 설정 후 시간: 0:0:0
```

예제 2 - Time 클래스(233쪽)

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

class Time {
private:
   int hour; // 0 - 23
   int minute; // 0 - 59
   int second; // 0 – 59

public:
   Time();   // 생성자
   Time(int h, int m, int s);
   void setTime(int h, int m, int s);
   void print();
};

// 첫번째 생성자
Time::Time() {
   setTime(0, 0, 0);
}
```

```
// 두번째 생성자
Time::Time(int h, int m, int s) {
   setTime(h, m, s);
}


// 시간 설정 함수
void Time::setTime(int h, int m, int s) {
   hour = ((h >= 0 && h < 24) ? h : 0);       // 시간 검증
   minute = ((m >= 0 && m < 60) ? m : 0); // 분 검증
   second = ((s >= 0 && s < 60) ? s : 0);    // 초 검증
}

// "시:분:초"의형식으로출력
void Time::print()  {
   cout << hour << ":" << minute << ":" << second << endl;
}
```

```
Time time1;
Time time2(13, 27, 6);
Time time3(99, 66, 77);
```

생성자에 잘못된 값 전달시 값 수정 가능을 보여줌

과제

- 과제 2개 → 6_1, 6_2 → DOOR의 수업활동 일지 에 각각 제출
- 과제 6_1 →
  - 245쪽 연습문제 2번(책 내용 약간 수정, 뒤에 설명) → P2()

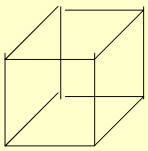
```
main{
        p2();       // 연습문제 2번 문제
}
```

- 과제 6_2 →
  - 사과장수 문제(뒤에 설명) → P3()

```
main{
        p3();       // 사과 장수
}
```

연습문제 2번 P2()

- 연습문제 2번 내용 수정 : Box 클래스
- 멤버변수, private
  - w, h, v_h 만 사용 (밑면 폭, 밑면 높이, 상자 높이, 부피 사용 안함)
- 멤버함수, public
  - empty (…) 함수 불필요
  - getVolume(…) : 상자의 부피를 계산하여 반환
  - print(…) : 상자의 밑면적, 상자 높이, 체적을 출력
  - 생성자 …
- 실행결과
  - main 에서 상자 3개 생성
  - 3개중 부피(체적) 가장 큰 것 출력



```
Microsoft Visual Studio 디버그 콘솔
========================================
박스 1 밑면적 : 6   상자 높이 : 4  부피 : 24
박스 2 밑면적 : 20   상자 높이 : 6  부피 : 120
박스 3 밑면적 : 1   상자 높이 : 1  부피 : 1
========================================
박스2의 부피가 가장 큽니다 : 120
```

연습문제 2번 P2()

```
int main() {
    Box b1(2, 3, 4), b2(4, 5, 6);
    Box b3;  // 매개변수 없는 경우 w=1, h=1, v_h=1 으로 지정

    cout << "========================================" << endl;
    cout << "박스 1 "; b1.print();
    cout << "박스 2 "; b2.print();
    cout << "박스 3 "; b3.print();
    cout << "========================================" << endl;
    int vb1 = b1.getVolume();
    int vb2 = b2.getVolume();
    int vb3 = b3.getVolume();

    // 3 개 상자의 부피 비교하여 가장 큰 것 출력하는 내용 작성

    return 0;
}
```

P3() 사과장수 문제

- 5 장의 은행 이체 문제와 유사
- 클래스 2개 필요 → 사과장수, 손님 class 필요

```
void P3(void)
{
    FruitSeller seller(1000, 20, 0);          // 상인

    FruitBuyer buyer(5000, 0);                // 손님

    buyer.BuyApples(seller, 2000);

    cout << "과일 판매자의 현황" << endl;
    seller.ShowSalesResult();

    cout << "과일 구매자의 현황" << endl;
    buyer.ShowBuyResult();
    return 0;
}
```

P3() 사과장수 문제

```
void P3(void)
{
    FruitSeller seller(1000, 20, 0);

    FruitBuyer buyer(5000, 0);

    buyer.BuyApples(seller, 2000);

    cout << "과일 판매자의 현황" << endl;
    seller.ShowSalesResult();

    cout << "과일 구매자의 현황" << endl;
    buyer.ShowBuyResult();
    return 0;
}
```

사과장수 생성 : 사과 가격 1000원, 사과 20개 소유, 소유금액 0원.

사과 구매자 생성 : 5000원 소유, 사과 0개 소유

아저씨 사과 2000원어치 주세요. → seller 는 사과 판매자

아저씨 오늘 얼마나 파셨어요.. 라는 질문의 대답

사과 구매하고 사과와 잔돈이 얼마야.. 라는 질문의 대답