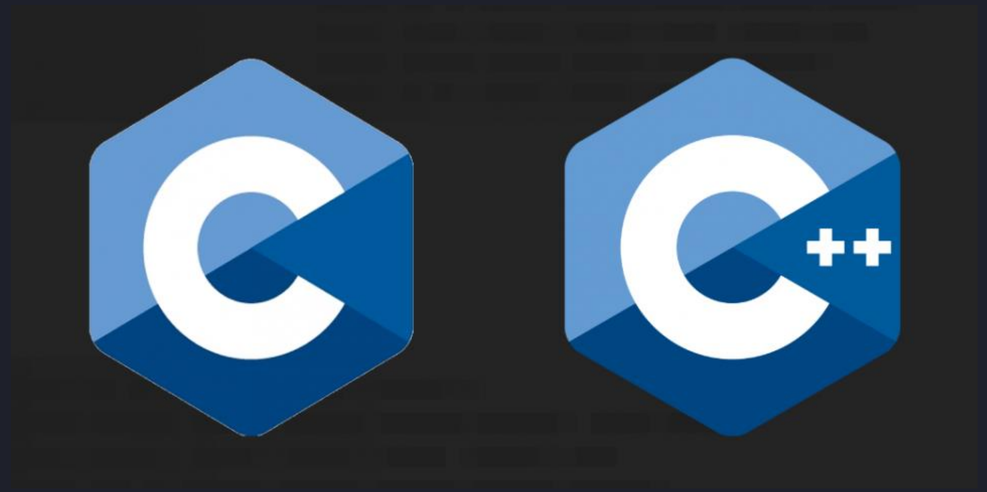

SPLIT 튜터링 8회차
:클래스

기계항공공학부 17학번 김기성

Contents

클래스의 필요성

- 가장 친숙한 라이브러리 중 하나인 vector가 없는 세상을 상상해봅시다.
- 하나 선언할 때마다 메모리 할당하고, 사이즈도 따로 저장해야하는 등 할 일이 정말 많습니다.

```
int* my_vector = new int[5]{1, 2, 3, 4, 5};
int my_vector_size = 5;

//my_vector를 변경하고 싶어짐
delete[] my_vector; // 메모리 할당 해제
my_vector = new int[6]{ 1, 2, 3, 4, 5, 6 }; // 새로운 메모리 영역 할당, 원소 대입
my_vector_size = 6;
```

```
vector<int> v = { 1, 2, 3, 4, 5 };
cout<< v.size() << endl;
v = { 1, 2, 3, 4, 5, 6 };
cout<<v.size()<<endl;
```

- 또다른 예시로 학적부를 관리하는 상황을 상상해봅시다.
- 학생과 관련된 데이터들을 한데 모아놓고, 원하는 방식대로만 조작할 수 있는 Student라는 자료형이 있다면 어떨까요?
- 두 개의 student 변수를 만들고, 각각에 학생에 관련된 정보를 저장할 수 있습니다.

```
string name1 = "gildong";
string student_id1 = "2023-11111";
int grade1 = 1;

string name2 = "cheolsu";
string student_id2 = "2022-12345";
int grade = 2;
```

```
student s1("gildong", "2023-11111", 1);
student s2("cheolsu", "2022-12345", 2);

cout << s1.name << endl;
cout << s1.student_id << endl;
cout << s1.grade << endl;

cout << s2.name << endl;
cout << s2.student_id << endl;
cout << s2.grade << endl;
```

```
gildong
2023-11111
1
cheolsu
2022-12345
2
```

클래스란?

- 클래스란, 데이터와 해당 데이터 조작에 필요한 함수를 묶어서 하나의 형판으로 만든 것입니다.
  - 문자열을 다루는 string, 배열을 다루는 vector 등도 하나의 클래스입니다.
  - 즉, 새로운 데이터 타입을 정의할 수 있게 됩니다.
- 클래스를 선언하는 방법은 다음과 같습니다.
- class <클래스명>{ // 클래스 내용 }; 형태로 선언합니다.
- 클래스 내부에는 변수, 함수를 선언할 수 있습니다.
  - private 영역
    - '.'연산자나 포인터에 '->'를 통해 접근할 수 없습니다.
    - 즉, 클래스 내부적으로 사용하는 함수나, 조작되어서는 안되는 변수를 이 영역에 선언합니다.
  - public 영역
    - '.'연산자나 포인터에 '->'를 통해 접근할 수 있습니다.
    - 외부에서 사용가능한 '인터페이스' 역할을 합니다.

```
class myClass {
// private 영역
private:
    int my_var; // 변수 선언

// public 영역
public:
    void set_my_var(int value) {
        my_var = value;
    }
    int get_my_var() {
        return my_var;
    }
};
```

객체 생성하기

- 기존의 데이터 선언방법과 같이 선언하면 됩니다.
- public 멤버 함수와 변수는 '.' 연산자나 '->'연산자로 접근할 수 있습니다.

```
myClass a;
cout << a.get_my_var() << endl;
a.set_my_var(5);
cout << a.get_my_var() << endl;
```

```
-858993460
5
```

- private 멤버 함수와 변수는 접근할 경우 에러가 발생합니다.

```
myClass a;
a.my_var;
```

C2248 'myClass::my_var': private 멤버('myClass' 클래스에서 선언)에 액세스할 수 없습니다.

- private 영역을 따로 만드는 이유: 데이터 은닉
  - 외부 코드에서 해당 변수에 접근할 수 없도록 합니다. 따라서 클래스 내부 변수가 다른 코드에 의해 직접 변경될 수 없고, public 함수로 규정된 방식으로만 조작될 수 있습니다.
- 기본값은 private입니다. 즉, public, private 여부를 지정하지 않는 경우 private 변수 혹은 함수가 됩니다.

생성자(constructor)

- 생성자(constructor)란?
  - 객체가 생성될 때 호출되는 함수입니다.
  - 주로 객체를 생성할 때 필요한 정보를 받아 내부 변수 초기화를 하는 역할을 합니다.
- 생성자를 선언하기 위해서는 return값을 명시하지 않고 클래스명과 같은 함수를 선언해주면 됩니다.
  - <클래스명>(매개변수1, 매개변수2, ...)
- 예시) vector와 같은 역할을 하는 my_vector 만들어보기
  - arr: my_vectdor 내부에서 실제로 원소를 담을 배열
  - size: 실제 원소의 개수
  - capacity: arr의 크기. 2의 배수(1, 2, 4, 8, ...)의 크기를 사용하며, size보다 크거나 같은 최초의 2의 배수가 됨
    - ex) size = 5 -> capacity = 8
  - STL vector에서 사용하는 기본 생성자와 두 개의 정수를 입력받는 생성자를 만들어봅시다.

```
vector<int> v1(); //비어있는 벡터 생성
vector<int> v2(5, 1); //크기 5, 모든 원소는 1로 초기화
```

vector 의 생성자

- 구현)

```
class my_vector {
private:
    int* arr;
    int size;
    int capacity;
public:
    //생성자 1: 아무 매개변수가 입력되지 않은 경우
    my_vector() {
        size = 0;
        capacity = 1;
        arr = new int[capacity];
    }
    //생성자 2: 정수 두개를 받은 경우 x가 입력된 size n의 배열을 만듦
    my_vector(int n, int x) {
        size = n;

        capacity = 1;
        while (capacity < size)
            capacity *= 2;

        arr = new int[capacity];
        for (int i = 0; i < size; ++i)
            arr[i] = x;
    }
};
```

```
my_vector v1();
my_vector v2(5, 0);
```

연산자 정의하기(연산자 오버로딩)

- 연산자 오버로딩이란, 연산자를 새로운 클래스에 대해 확장하여 적용하는 것을 의미합니다.
- C++에는 오버로딩이 가능한 다양한 연산자들이 있습니다.
  - 각 연산자는 몇 개의 파라미터를 받을지 정해져있습니다.
  - my_vector에 대해서 [] 연산자를 구현해봅시다.
- 연산자를 정의하는 법은 함수와 매우 유사합니다.
  - 함수 이름을 operator<연산자 이름>으로 정의하고, 매개변수와 return 형식을 정해주면 끝입니다.
  - 매개변수의 개수는 정해져있습니다. 예를 들어 +연산자는 두 개의 입력을 받고, ++연산자는 한 개의 입력을 받는 것입니다.
  - 그런데 매개변수 중 첫번째 매개변수는 해당 클래스의 객체로 정해져 있습니다.

```
// my_vector 클래스의 public 영역
int& operator[](int idx) {
    return arr[idx];
}
```

```
cout << v2[3] << endl;
```

```
0
```

  - return 형식에 &을 붙이면 복사본이 아니라 원본을 반환하므로, v2[3] = 1;과 같은 방법으로 원소 변경이 가능해집니다.

소멸자 정의하기

- 소멸자는 객체가 소멸할 때(=변수의 수명이 끝날 때) 호출되는 함수입니다.
- 소멸자 정의 방법: ~<클래스 이름>()
- 소멸자에서 주로 해야 하는 작업은, 동적할당된 메모리를 해제해주는 것입니다.
- 클래스 내의 지역 변수는 소멸자가 끝날때 사라지지만, 동적할당된 메모리는 자동으로 해제되지 않습니다.

```
~my_vector() {
    delete[] arr;
}
```

여담: 내부 함수 정의를 밖으로 빼내기

- 클래스 내부에서 바로 함수나 연산자를 정의해도 되지만, 클래스를 정의할 때 함수 구현을 함께 하는 것은 가독성을 해칩니다.
- 그래서 클래스 정의와 내부 함수 구현을 분리하는 경우가 많습니다.
  - 클래스 외부에서 <자료형> <클래스명>::<함수명>(매개변수1, ...) 형태로 작성하면 됩니다.
- 정의와 구현을 한 파일에 해도 되지만, 클래스와 구현을 다른 파일에 하는 경우가 많습니다.

```
class my_vector {
private:
    int* arr;
    int size;
    int capacity;
public:

    //생성자 1: 아무 매개변수가 입력되지 않은 경우
    my_vector() {
        size = 0;
        capacity = 1;
        arr = new int[capacity];
    }
    //생성자 2: 정수 두개를 받은 경우 x가 입력된 size n의 배열을 만듦
    my_vector(int n, int x) {
        size = n;

        capacity = 1;
        while (capacity < size)
            capacity *= 2;

        arr = new int[capacity];
        for (int i = 0; i < size; ++i)
            arr[i] = x;
    }

    ~my_vector() {
        delete[] arr;
    }

    // my_vector 클래스의 public 영역
    int& operator[](int idx) {
        return arr[idx];
    }
};
```

```
class my_vector {
private:
    int* arr;
    int size;
    int capacity;
public:
    my_vector();
    my_vector(int n, int x);
    ~my_vector();
    int& operator[](int idx);
};
```

```
my_vector::my_vector() {
    size = 0;
    capacity = 1;
    arr = new int[capacity];
}
my_vector::my_vector(int n, int x) {
    size = n;

    capacity = 1;
    while (capacity < size)
        capacity *= 2;

    arr = new int[capacity];
    for (int i = 0; i < size; ++i)
        arr[i] = x;
}

my_vector::~my_vector() {
    delete[] arr;
}

int& my_vector::operator[](int idx) {
    return arr[idx];
}
```

여담: 생성자에 대한 TMI

- 아무 매개변수를 받지 않는 생성자를 기본 생성자(default constructor)라고 합니다.
  - 기본 생성자는 필수적으로 정의되어야 합니다.
  - 아무 생성자를 정의하지 않은 경우 비어있는 기본생성자가 자동으로 생성되지만, 매개변수를 받는 다른 생성자를 정의한 경우에는 기본 생성자가 기본적으로 생성되지 않습니다.

```
class my_vector {
public:
    int* arr;
    int size;
    int capacity;
public:
    //my_vector();
    my_vector(int n, int x);
    ~my_vector();
    int& operator[](int idx);
};
```

C2512 'my_vector': 사용할 수 있는 적절한 기본 생성자가 없습니다.

- 같은 클래스의 객체를 받는 생성자를 복사 생성자(copy constructor)라고 합니다. `my_vector(my_vector& v);`
  - 복사 생성자는 = 연산자를 통해 다른 객체에 할당하거나, 함수에 매개변수로 전달할때, 반환값으로 받을 때 호출됩니다(단, pass by reference일 때는 호출되지 않겠죠?)
  - 복사 생성자도 따로 정의해주지 않으면 기본적으로 생성됩니다.
  - 그런데, 이 경우 의도하지 않은 동작이 발생할 수 있습니다.

여담: 생성자에 대한 TMI

- 올바르지 않은 복사 생성자

```
my_vector::my_vector(my_vector& v) {
    arr = v.arr;
    size = v.size;
    capacity = v.capacity;
}
```

  - 이유 1: 의도한 대로 동작하지 않습니다. v1 = v2를 한 후 v1의 원소를 변경하면 v2도 바뀝니다.
  - 이유 2: 두 my_vector의 소멸자가 호출되면 같은 주소에 delete[]가 두 번 되게 됩니다.
  - 그런데 기본적으로 생성된 복사생성자는 이 방식으로 구현되어 있습니다.
- 올바른 복사 생성자

```
my_vector::my_vector(my_vector& v) {
    arr = new int[v.capacity];
    for (int i = 0; i < v.size; ++i) {
        arr[i] = v[i];
    }
    size = v.size;
    capacity = v.capacity;
}
```

연습문제

- 연습문제 1) vector의 size()함수를 구현해봅시다.
  - size를 public 변수로 만들었다면 함수가 없어도 되지 않을까요? 왜 굳이 함수로 구현했을까요?
- 연습문제 2)
  - vector 클래스의 push_back함수는 int 하나를 받아 원소를 맨 뒤에 추가합니다. 이 함수를 구현해봅시다.
  - 주의할 점) capacity랑 size가 같은 경우에 push_back을 호출한 경우에는 capacity를 두 배로 늘리고, 해당 크기의 정수 배열을 새로 동적할당한 다음, 원래 배열의 원소를 모두 새 배열에 복사 붙여넣기해야합니다.
- 연습문제 3)
  - C++ 에서 vector 끼리의 덧셈은 정의되어있지 않습니다.
  - 그런데 파이썬에서 vector와 비슷한 기능을 하는 list에는, list끼리 더할 경우에 두 list가 이어지도록 구현되어있습니다. 즉, {1, 2, 3}+{4, 5, 6}을 수행할 경우 {1, 2, 3, 4, 5, 6}을 담고 있는 list가 반환됩니다.
  - operator+연산자를 오버로드하여 해당 연산자를 구현해봅시다.