24. C++의 구조체(structure)

• 구조체는 다른 형(type)의 변수 여러 개를 필드(field)로 선언하여, 한 개의 대표이름을 통하여 참조할 수 있습니다.
• 링크드 리스트(linked list) 같은 자료구조data structure를 구현하는 데에는 구조체는 필수적인 역할을 합니다.

• 구조체는 **변수의 선언(variable declaration)**뿐만 아니라, **형의 정의(type definition)**도 할 수 있다는 것입니다.

• 만약 우리가 회원 관리 프로그램을 만들기 위해, 회원의 이름과 나이가 필요하다고 합시다. 또한 이름은 영문자 20자를 넘지 않는다고 가정하고, 회원의 수가 100명이 넘지 않는다는 가정을 합시다.

```
short age[100];
char name[100][20];
```

• 필요한 변수가 더 많아진다면, 이러한 변수의 선언이 더 많아질 것이고, 많아진 변수로 인해 굉장히 골머리를 알아야 할 것입니다.
• 회원의 수를 예측하기 어렵다면, 고정된 길이 100의 배열을 선언해서는 문제를 해결할 수 없다는 것입니다.

• 해결책은 바로 구조체(링크드리스트)를 이용하는 것입니다. 서로 다른 형 – short와 char [] - 을 레코드record의 필드로 취급하는 것입니다.
• short와 char [20]을 필드로 가지는 구조체의 형은 다음과 같이 작성을 시작할 수 있습니다.

```
struct {
      short age;
      char name[20];
}
```

• 이것이 '구조체 형'이라는 점에 주목해야 합니다.

```
struct { short age; char name[20]; } i;
①                                   ②
```

• ①은 형 이름에 해당하는 부분이며, ②는 변수이름에 해당합니다.

- 구조체 i에 대해 다음과 같은 질문을 할 수 있습니다.

  (1) i가 의미하는 것은 무엇일까요?
  (2) 필드 age와 name은 어떻게 참조할 수 있을까요?

- i는 **구조체 자신(structure itself)**입니다.
- 구조체의 필드 - 앞으로 필드를 **멤버(member)**라고 부르기로 합시다 - 를 참조하기 위해, 멤버 **참조 연산자(member reference operator: . 와 ->)**를 사용합니다.
- 그러므로 이름이 "seojt"인 30살의 회원은 다음과 같이 정보를 초기화할 수 있습니다.

```
i.age=30;
strcpy(i.name,"seojt");
```

```
#include <stdio.h>
#include <string.h>

void main() {
    struct {
        short age;
        char name[20];
    } i;

    i.age=30;
    strcpy(i.name,"seojt");
    printf("%d,%s\n",i.age,i.name);
    //      30,seojt
}
```

• 구조체는 형 정의의 기능을 가지고 있습니다.
• 위에서 사용한 구조체를 SMan이라는 형으로 정의하려면, struct와 첫 번째 여는 브레이스(open brace; {)사이에 사용하려는 형 이름을 적어 줍니다.

```
struct SMan {
    short age;
    char name[20];
};
```

• 위 문장이 변수를 선언한 것이 아니라, SMan이라는 형을 정의한 것임에 유의하세요. 문장이므로 끝에 세미콜론(;)이 필요합니다.
• C에서는 struct SMan이 형 이름입니다. 그러므로 C언어에서는 SMan을 **형 이름(type name)**이라고 하지 않고, **꼬리표 이름(tag name)**이라고 했습니다.
• C++에서는 struct SMan도 형이고, SMan도 형입니다. 그러므로 C++에서는 SMan을 주로 형 이름으로 사용합니다.

```
#include <stdio.h>
#include <string.h>

struct SMan {
    short age;
    char name[20];
};

void main() {
    SMan i;//C에서는 struct SMan i; 라고만 해야 한다.

    i.age=30;
    strcpy(i.name,"seojt");
    printf("%d,%s\n",i.age,i.name);
    //      30,seojt
}
```

• 아래 문장은 SMan이라는 형을 정의하면서, SMan형의 구조체 변수 i,j를 선언하고 있습니다.

```
struct SMan {
    short age;
    char name[20];
} i,j;
```

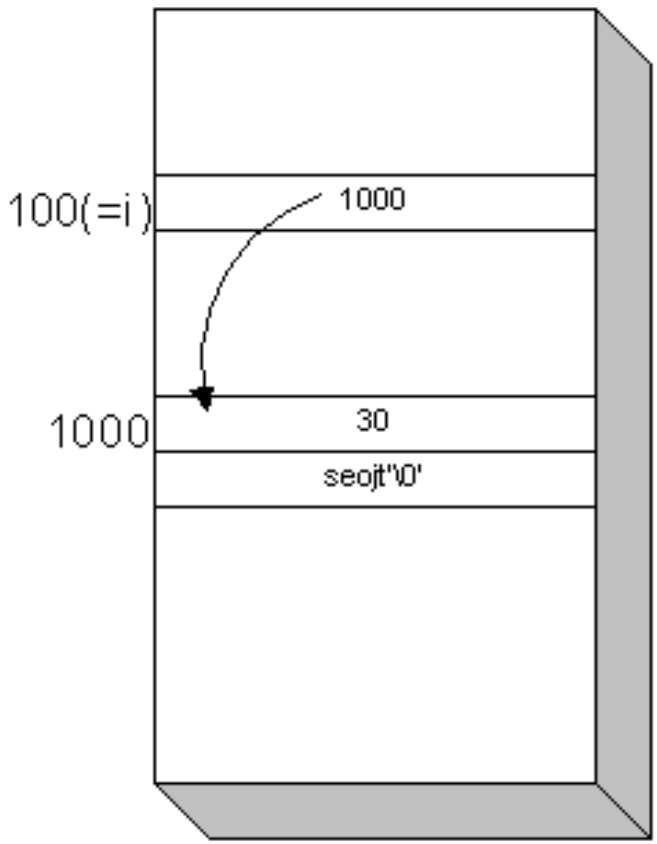
• 아래와 같이 선언된 구조체 포인터 변수를 통해 멤버는 어떻게 접근할 수 있을까요?

```
SMan *i;
```

• i는 포인터이지만, i가 가리키는 내용, *i는 구조체입니다. 그러므로, * 연산자를 이용해서 구조체를 접근한 다음 . 을 사용해서 구조체의 멤버를 접근하는 것이 가능합니다.

```
(*i).age=30;
strcpy((*i).name,"seojt");
```

```
#include <stdio.h>
#include <string.h>

struct SMan {
    short age;
    char name[20];
};

void main() {
    SMan *i;

    (*i).age=30;
    strcpy((*i).name,"seojt");
    printf("%d,%s\n",(*i).age,(*i).name);
    //      30,seojt NO! Run Time Error!
}
```



할당하지 않은 구조체를 건드림: i에 우연히 1000이 들어있었다고 합시다. 위 프로그램은 할당되지 않은 1000~1021번지 사이에 값을 넣고 있습니다.

• 우리는 C를 사용할 때는 다음과 같이 표준함수를 이용해서 메모리 할당을 했습니다.

```
i = (SMan*)malloc(22);
```

• 구조체의 크기 계산이 복잡한 경우는 다음과 같이 문장을 작성할 수도 있으며, 이것이 더 일반적인 방법입니다.

```
i=(SMan*)malloc(sizeof SMan);
```

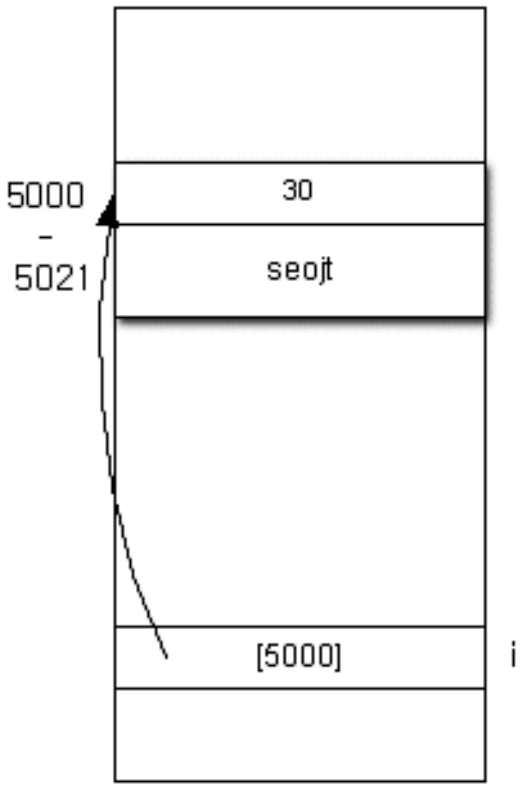
• malloc()은 함수이므로, 선언되어야 합니다. 그러므로 다음 문장이 소스의 첫 부분에 필요합니다.

```
#include <stdlib.h>
```

• 할당한 메모리는 반드시 해제해 주어야 합니다.

```
free(i);
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct SMan {
    short age;
    char name[20];
};

void main() {
    SMan *i;

    i=(SMan*)malloc(sizeof SMan);
    (*i).age=30;
    strcpy((*i).name,"seojt");
    printf("%d,%s\n",(*i).age,(*i).name);
    //      30,seojt
    free(i);
}
```



구조체의 할당: 동적으로 할당한 메모리가 [5000]이라면, 구조체를 위한 메모리는 [5000]-[5021]에 걸쳐 22바이트가 할당될 것입니다. 이곳을 i가 가리키게 됩니다. i는 구조체가 아니라, *i가 구조체라는 사실을 주의하세요.

```
#include <stdio.h>
#include <string.h>
//#include <stdlib.h> 이 문장은 더 이상 필요없다.

struct SMan {
    int age;
    char name[20];
};

void main() {
    SMan *i;

    i=new SMan;
    (*i).age=30;
    strcpy((*i).name,"seojt");
    printf("%d,%s\n",(*i).age,(*i).name);
    //      30,seojt
    delete i;
}
```

• 구조체는 구조체 포인터로 사용할 일이 훨씬 많습니다. 그래서 구조체의 멤버를 쉽게 참조하기 위해서 두 번째의 구조체 멤버 참조 연산자인 **화살표 연산자(arrow operator: ->)**의 사용이 가능합니다.

• (*i).age는 i->age로 사용할 수 있습니다. 이 연산자의 새로운 기능은 없습니다. 단지, 편리한 입력을 위해서 사용될 뿐입니다.

```
#include <stdio.h>
#include <string.h>

struct SMan {
    int age;
    char name[20];
};

void main() {
    SMan *i;

    i=new SMan;
    i->age=30;
    strcpy(i->name,"seojt");
    printf("%d,%s\n",i->age,i->name);
    //      30,seojt
    delete i;
}
```

좀 더 자연스러운 구조체

• 위의 구조체를 일반화하기 위하여, 구조체의 멤버 변수를 조작하는 함수 등 구조체와 관계된 함수를 3개 만들기로 했습니다.

• 이들 함수는 나이를 정하고, 이름을 정하고, 나이와 이름을 출력하는 일 등을 할 것입니다.

```
#include <stdio.h>
#include <string.h>

struct SMan {
    int age;
    char name[20];
};

void SetAge(SMan s,int a) {
    s.age=a;
}
```

```
void SetName(SMan s,char n[]) {
    strcpy(s.name,n);
}

void Print(SMan s) {
    printf("age=%d\n"
           "name=%s\n",
           s.age,s.name);
}

void main() {
    SMan i;

    SetAge(i,30);
    SetName(i,"seojt");
    Print(i);
}
```

• 일반적으로 구조체를 함수의 파라미터로 전달할 때는 주소를 전달합니다. C++에서는 참조(reference)가 좋은 해결책이 될 수도 있습니다.
• 그럼에도 불구하고 여전히 문제점이 존재합니다.
• 이 문제점은 문법적인(syntax), 논리적인(semantic) 에러도 아닙니다. 단지 문제 그 자체와 관련된 좀 더 상위 레벨의 **설계 개념의 위반**입니다.

① 세 개의 함수는 구조체를 조작하기 위해, 매번 구조체의 주소를 받아야 합니다.
② 세 개의 함수는 구조체 SMan과 상관되어 있습니다. 하지만, 이 세 개의 함수가 SMan과 상관되어 있다는 것을 어떻게 알 수 있을까요?

• ②번 문제는 C의 설명문(comment)으로 어느 정도 처리할 수는 있지만, 완벽한 것은 아닙니다.
• 언어 자체의 문법에 의해, 그러한 설계의 개념이 표현될 수 있으면 구조체가 강력해질 것입니다.

```
#include <stdio.h>
#include <string.h>

/*---------------- start of struct SMan ------------------*/
struct SMan {
    int age;
    char name[20];
};

void SetAge(SMan &s,int a) {
    s.age=a;
}

void SetName(SMan &s,char n[]) {
    strcpy(s.name,n);
}

void Print(SMan &s) {
    printf("age=%d\n"
```

```
            "name=%s\n",
            s.age,s.name);
}
/*---------------- end of struct SMan -------------------*/

void main() {
    SMan i;

    SetAge(i,30);
    SetName(i,"seojt");
    Print(i);
}
```

• 문제점에 대한 핵심 질문은 다음과 같습니다.

"왜 구조체와 구조체를 조작하는 함수를 별개로 구현해야 하는가?"

- 우리가 존재하는 실제 세계의 **대상물(object)**■ - **객체** - 을 생각해 봅시다.

C++에서는 **객체**(object)라고 합니다. 객체는 실세계의 대상물을 의미하며, 자연 세계가 그러하듯이 객체는 정적인(static) 특성과 동적인(dynamic) 특성을 가집니다. **클래스**는 이러한 객체를 모델링하는 강력한 도구(tool)입니다. 정적인 특성은 멤버 변수로, 동적인 특성은 멤버 함수로 모델링합니다. 즉 C++의 문법에 의해 설계 개념이 지원되는 것입니다!

- 이러한 객체는 C의 구조체로 표현이 불가능합니다.
- 예를 들어 자동차라는 객체를 시뮬레이션하는 프로그램을 코딩한다고 가정해 봅시다. 자동차를 모델링하기 위해, 자동차의 정적인 특성 – 색, 메이커, 가격 등 – 만을 구조체로 모델링할 수 있습니다.
- 하지만, 이 구조체와 관련된 자동차의 동적인 특성 - 운전한다, 가속기(accelerator)를 밟는다, 화면에 그린다, 등 - 은 구조체를 함수의 파라미터로 받는 함수로 구현하여야 합니다.
- 왜 구조체와 관련된 함수를 별개로 구현하여야 하는가요?

```
struct SMan {
    int age;
    char name[20];
    void SetAge(int a);//함수는 동적인 특성을 표현한다
    void SetName(char n[]);
    void Print();
};
```

• SetAge()등의 함수의 첫번째 파리미터로 구조체의 참조가 사라진 것에 주의하세요.

• 이 멤버 함수들은 구조체 자신의 멤버 변수를 접근하므로 더 이상 구조체를 파라미터로 받을 필요는 없습니다.

• 위 정의에는 멤버 함수의 선언이 있을 뿐입니다. 이 멤버 함수들을 정의하여야 합니다. 그래서 아래와 같이 정의를 추가하였습니다.

```
#include <stdio.h>
#include <string.h>

struct SMan {
    int age;
    char name[20];
    void SetAge(int a);
    void SetName(char n[]);
    void Print();
};

void SetAge(int a) {
    age=a;//age는 도대체 어디에 할당된 것인가?
}

void SetName(char n[]) {
```

```
    strcpy(name,n);
}

void Print() {
    printf("age=%d\n"
           "name=%s\n",
           age,name);
}

void main() {
    SMan i;

    i.SetAge(30);
    i.SetName("seojt");
    i.Print();
}
```

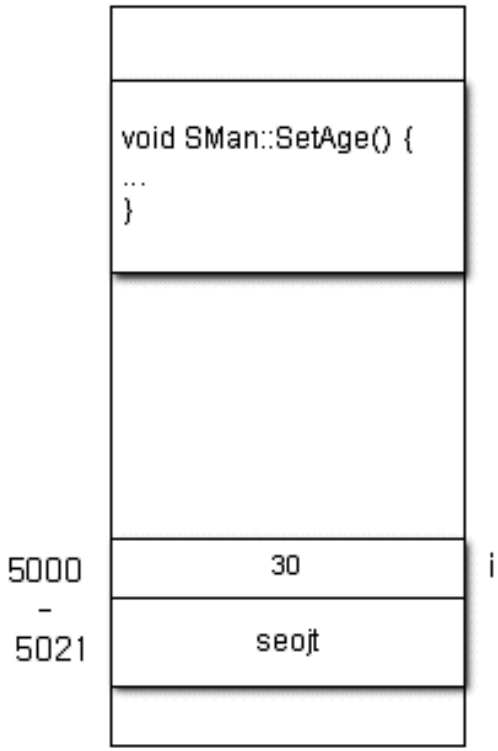
범위 해결사

- 위 프로그램에는 여전히 문법적인 에러가 존재합니다.
- SetAge()는 멤버 함수이지 함수가 아닙니다.

- SetAge()등이 SMan의 멤버 함수라는 것을 문법적으로 표현해 줄 수 있는 방법이 있어야 합니다.
- 이것을 위해서 C++에 새로 도입된 연산자인 **범위 해결사(scope resolver: ::)**를 사용하여야 합니다.

```
#include <stdio.h>
#include <string.h>

struct SMan {
    int age;
    char name[20];
    void SetAge(int a);
    void SetName(char n[]);
    void Print();
};

void SMan::SetAge(int a) {
    age=a;//age는 SMan의 멤버 변수이다
}

void SMan::SetName(char n[]) {
    strcpy(name,n);
}
```

```
void SMan::Print() {
    printf("age=%d\n"
           "name=%s\n",
           age,name);
}

void main() {
    SMan i;

    i.SetAge(30);
    i.SetName("seojt");
    i.Print();
}
```



멤버 함수의 존재: 프로그램이 실행되었을 때, 멤버 함수는 객체와는 상관없이 메모리에 항상 존재합니다. 객체가 할당될 때, 멤버 변수만을 위한 메모리 공간이 할당될 뿐입니다. 그러므로, 멤버 함수의 호출은 컴파일 시간에 결정됩니다. 컴파일러는 정확하게 멤버 함수가 몇 번지에 있는지 계산할 수 있습니다.

• SMan 구조체의 멤버 함수 선언부분에는 SMan::이 없어도 될까요? 범위에 의해 컴파일러가 검사할 수 있으므로 없어도 됩니다.
• 하지만 없는 것이 아니라, 자동으로 붙여지는 것입니다. 그러므로 명시적으로 SMan::을 적어주어도 아무 상관이 없습니다.

```
struct SMan {
    int age;
    char name[20];
    void SMan::SetAge(int a);
    void SMan::SetName(char n[]);
    void SMan::Print();
};
```

• 이제 우리는 C++의 변경된 구조체를 사용하여 실세계의 객체를 모델링한 것입니다!

- struct 대신에 class를 사용해도 된다는 말일까요?
- 그렇습니다. 다만 class에서 사용하는 몇 가지 규칙 때문에, 몇 가지 변경이 필요합니다.

```
#include <stdio.h>
#include <string.h>

class SMan {
  private://이 후의 멤버들의 접근 권한이 private임을 선언한다.
    int age;
    char name[20];
  public://이 후의 멤버들의 접근 권한이 public임을 선언한다.
    void SetAge(int a);
    void SetName(char n[]);
    void Print();
};

void SMan::SetAge(int a) {

    age=a;
```

```
}

void SMan::SetName(char n[]) {
    strcpy(name,n);
}

void SMan::Print() {
    printf("age=%d\n"
           "name=%s\n",
           age,name);
}

void main() {
    SMan i;

    i.SetAge(30);
    i.SetName("seojt");
    i.Print();
}
```

• SMan형의 변수 i를 선언했을 때, SMan을 **형(type)**, i를 **변수(variable)**라고 하는 것은 어색합니다.
• i는 더 이상 변수가 아닙니다. 변수는 함수를 가질 수 없습니다.
• 그러므로 클래스에대해서, SMan을 클래스(class), i를 **객체(object)**라고 합니다.
• 그리고 이러한 것을 '변수를 선언한다'라고 하지 않고, '**클래스 SMan의 객체 i를 만든다**'고 합니다.
• 객체를 만드는 과정을 '**인스턴스화(instantiation)**'라고 합니다. 인스턴스화 된 객체를 인스턴스**(instance)**■라고 합니다.
• 클래스 SMan의 입장에서 i는 자신의 인스턴스(instance)입니다. 하지만, 객체 i의 입장에서 자신은 SMan의 객체(object)입니다.

• 회원이 1명 증가할 때마다, 우리가 해야 할 일은 SMan의 객체를 하나 더 만드는 일을 하면 됩니다.

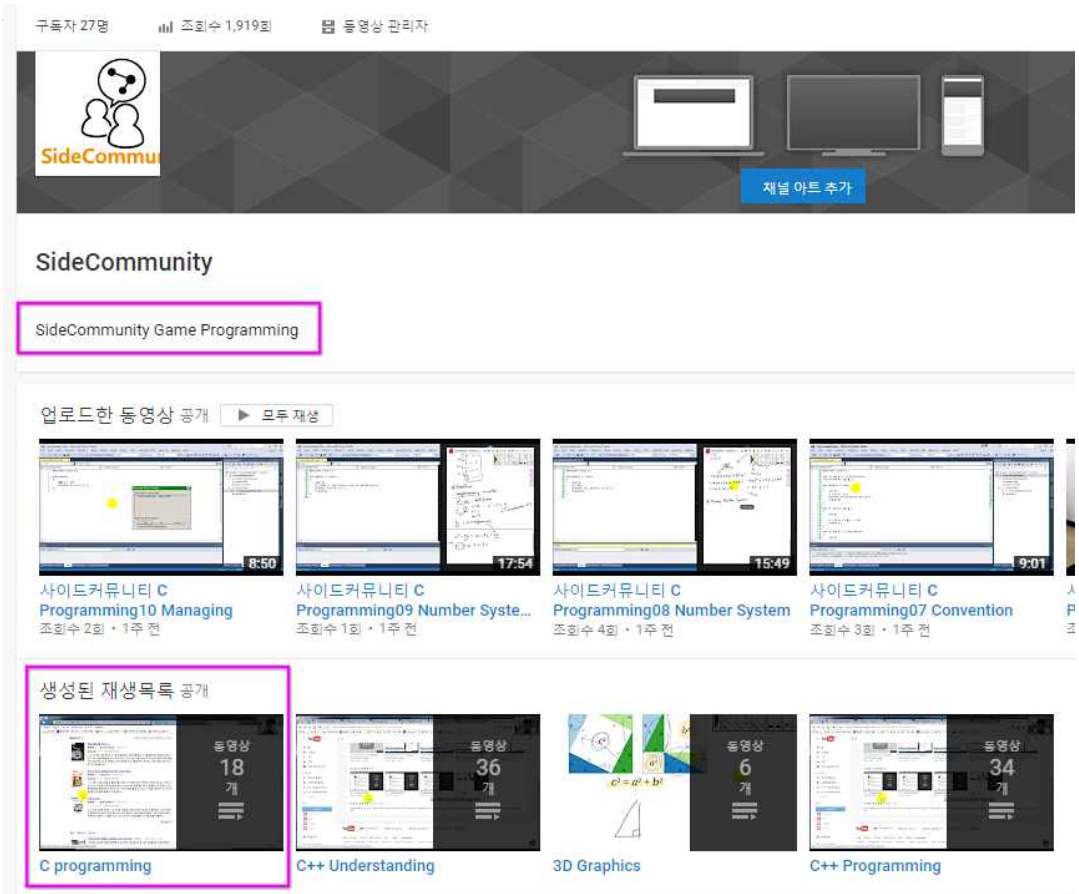
…

```
void main() {
    SMan i;

    i.SetAge(30);
    i.SetName("seojt");
    i.Print();

    SMan j;
    j.SetAge(0);
    j.SetName("baby");
    j.Print();
}
```

C++을 배우려는 독자들에게

• C와 C++ 그리고 C++의 고급 주제들에 대한 저자의 동영상 강의를 유튜브에서 시청할 수 있습니다.
• 동영상 강좌는 sidecommunity game programm채널에 준비되어 있습니다.

https://www.youtube.com/playlist?list=PLrrTotxaO6kjxlHovqDNTdSFZ8tcAvQkO



Youtube의 SideCommunity Game Programming 채널: 책의 C 강좌에 해당하는 내용을 동영상으로 시청할 수 있습니다.

Ritchie가 말하기를 C가 있으라, 그러자 C가 있었다. 그것이 사람들이 보기에 좋았다. 하지만, C가 독처하는 것은 좋지 않았다. 후에 Bjarne이 생각했다. '내가 C를 돕는 베필을 지으리라.' 그래서 C의 형상을 따라, 클래스(class)를 만들었다.

실습문제

1. 구조체(structure)는 모든 멤버가 public인 클래스와 동일한가요?

2. 공용체는 멤버 함수를 가질 수 있는가요?

3. 클래스가 객체를 기술하는 틀이지만, 객체간의 관계를 나타낼 수는 없습니다. 객체간의 관계를 나타내기 위한 방법은 무엇인가요?

@