

구조체(struct)

❖ 기존 타입의 변수들의 조합으로서 새로운 타입을 정의

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;

struct Student {
    string name ;
    Grade grade ;
    string phoneNumber ;
};
```

구조체 변수/객체의 생성 및 초기화

❖ 구조체 객체의 생성 및 초기화

```
Student st1 = {"Park", FRESH, "000-0000"} ;
```

```
Student* st2 = new Student ;
```

❖ 동일 구조체 객체에 의한 초기화 및 대입이 가능

초기화	실제 동작
Student st3 = st1 ; 또는 Student st3(st1) ;	Student st3 ; st3.name = st1.name ; st3.grade = st1.grade ; st3.phoneNumber = st1.phoneNumber ;

구조체 멤버 필드의 접근

- ❖ 객체.필드
- ❖ 객체포인터->필드

```
// 개별 필드의 접근  
st2->name = st1.name ;  
st2->grade = st1.grade ;  
st2->phoneNumber = st1.phoneNumber ;
```

구조체간의 대입

- ❖ C++ 언어에서는 구조체 간에 대입문이 가능함
- ❖ 구조체를 구성하는 개별 멤버 필드간의 대입과 동일

대입	실제 동작
<code>st3 = st1 ;</code>	<code>st3.name = st1.name ;</code> <code>st3.grade = st1.grade ;</code> <code>st3.phoneNumber = st1.phoneNumber ;</code>

C 언어와 C++ 언어의 구조체

	C ++ 언어	C 언어
정의 방법	struct Rectangle { ... } ;	
구조체변수 정의	Rectangle r1, r2 ;	struct Rectangle r1, r2 ;
구조체간의 대입 예) r2 = r1 ;	허용	불허
함수의 매개변수 int getArea(Rectangle)	허용	불허 int getArea(struct* Rectangle) 는 허용
함수의 반환 값 Rectangle getRect()	허용	불허 Rectangle* getRect()는 허용

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
```

```
// 구조체 타입의 정의
```

```
struct Student {  
    string name ;  
    Grade grade ;  
    string phoneNumber ;  
};
```

```
int main() {
```

```
    // 개별 필드의 초기화
```

```
    Student st1 = {"Park", FRESH, "000-0000"} ;
```

```
    // 개별 필드의 접근
```

```
    Student* st2 = new Student ;
```

```
    st2->name = st1.name ;
```

```
    st2->grade = st1.grade ;
```

```
    st2->phoneNumber = st1.phoneNumber ;
```

```
    delete st2 ;
```

```
    // 구조체 초기화
```

```
    Student st3 = st1 ; // 또는 Student st3(st1) ;
```

```
    // 구조체간의 대입
```

```
    st3 = st1 ;
```

```
    /* 아래와 같이 개별 필드 별로 대입하는 것과 유사함
```

```
    st3.name = st1.name ;
```

```
    st3.grade = st1.grade ;
```

```
    st3.phoneNumber = st3.phoneNumber ;
```

```
    */
```

```
    // 개별 필드의 접근
```

```
    cout << st3.name << '\t' << st3.grade << '\t' << st3.phoneNumber << endl ;
```

```
}
```

구조체 타입을 std::vector에 추가하기 1

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
std::istream& operator>>( std::istream& is, Grade& grade ){
    int igrade ;
    if ( is >> igrade )
        grade = static_cast<Grade>( igrade ) ;
    return is ;
}

int main() {
    int count=0;
    std::cin >> count;

    std::vector<Student> vec;
    for (int i=0; i<count; ++i) {
        Student st;
        std::cin >> st.name; std::cin >> st.grade; std::cin>>s1.phoneNumber;
        vec.push_back(st);                                //복사(copy)
        //vec.push_back(std::move(st));                  //이동(move)
    }

    for(auto& it : vec) {
        std::cout << it.name << "Wt" << it.grade << "Wt" << it.phoneNumber << std::endl;
    }
}
```

구조체 타입을 std::vector에 추가하기 2

```
int main() {
    int count=0;
    std::cin >> count;

    std::vector<Student*> vec;
    for (int i=0; i<count; ++i) {
        //포인터 - new를 이용해 Student 객체를 메모리(heap)에 생성
        Student* pst = new Student;
        std::cin >> pst->name >> pst->grade >> pst->phoneNumber;
        vec.push_back(pst);                //복사(copy)
        //vec.push_back(std::move(pst));    //이동(move)
    }

    for(auto& it : vec) {
        std::cout << it->name << "Wt" << it->grade << "Wt";
        std::cout << it->phoneNumber << std::endl;
    }
    //delete를 이용해 Student 객체를 메모리(heap)에서 삭제
    for(auto& it : vec)
        delete it;
}
```


구조체 타입을 std::vector에 추가하기 3

```
#include <memory>

int main() {
    int count=0;
    std::cin >> count;

    std::vector<std::unique_ptr<Student>> vec;
    for (int i=0; i<count; ++i) {
        //스마트 포인터
        std::unique_ptr<Student> pStudent(new Student);
        std::cin >> pStudent->name >> pStudent->grade >> pStudent->phoneNumber;
        //vec.push_back(pStudent);           //복사(copy) - 에러
        vec.push_back(std::move(pStudent)); //이동(move)
    }

    //for(auto it: vec)                       //복사(copy) - 에러
    for(auto& it : vec) {
        std::cout << it->name << "₩t" << it->grade << "₩t";
        std::cout << it->phoneNumber << std::endl;
    }
}
```


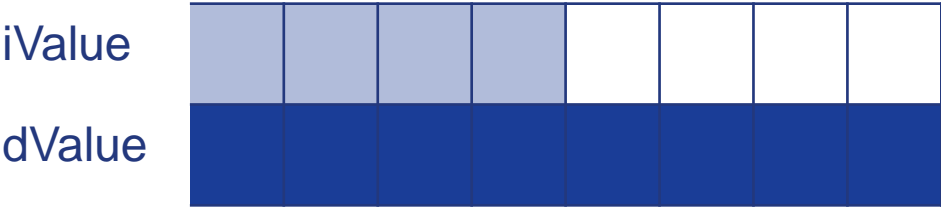
공용체(union)

❖ 동일한 공간을 공유하는 여러 개의 변수를 정의

```
union NumericUnion
{
    int      iValue;
    double   dValue;
};
```

예제 프로그램	실행 예
<pre>int main() { NumericUnion values ; values.iValue = 10 ; cout << values.iValue << endl ; values.dValue = 3.14F ; cout << values.dValue << endl ; }</pre>	<pre>// 10 // 3.14</pre>

공용체와 구조체

<pre>struct NumericStruct { int iValue; double dValue; };</pre>	 <p>0 4 8 12</p>
(a) 구조체의 메모리 구성	
<pre>union NumericUnion { int iValue; double dValue; };</pre>	 <p>0 4 8</p>
(b) 공용체의 메모리 구성	

Good Design: 공용체의 사용은 자제

- ❖ 공용체는 복수 개의 변수가 동일한 메모리를 사용하므로 사용되는 메모리의 양을 줄일 수 있다는 측면에서 잇점
- ❖ 동일한 메모리를 사용하고 있으므로 공용체의 각 필드의 값은 서로 영향을 미치게 된다

예제 프로그램	실행 예
<pre>int main() { NumericUnion values ; values.iValue = 10 ; cout << values.iValue << endl ; cout << values.dValue << endl ; values.dValue = 3.14F ; cout << values.iValue << endl ; cout << values.dValue << endl ; }</pre>	<pre>// 10 // 1.4013e-044 // 1078523331 // 3.14</pre>

typedef

- ❖ 기존의 타입과 동일한 역할을 하는 새로운 타입 별칭(alias)을 정의

```
typedef unsigned int Age ;  
typedef double celsius_t ;  
typedef double fahrenheit_t;
```

- ❖ 새롭게 정의된 타입 이름은 변수의 정의, 함수의 매개변수 타입 등으로 동일하게 사용

```
Age me = 20 ;  
celsius_t getCelsius(fahrenheit_t f);
```

- ❖ 참고

```
using Age = unsigned int;
```

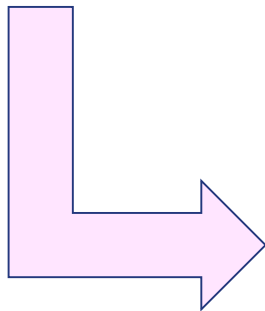
typedef의 활용 예 in C++ STL

basic_string.h	<pre>typedef typename _Alloc_traits::pointer pointer; typedef __gnu_cxx::__normal_iterator<pointer, basic_string> iterator; typedef __gnu_cxx::__normal_iterator<const_pointer, basic_string> const_iterator;</pre>
사용	<pre>... for(string::iterator it=str.begin(); it!=str.end(); ++it) ... for(string::const_iterator it=str.cbegin(); it!=str.cend(); ++it)</pre>

Good Design: 새로운 타입에 범위를 설정하고 싶다면 클래스를 사용

- ❖ typedef는 타입이 취할 수 있는 값에 대한 범위를 제한하지는 못한다

```
typedef unsigned int Age ;  
Age a ;  
a = -10 ;
```



```
const int INVALID_AGE = -1 ;  
class Age {  
    int age ;  
public:  
    Age(const int a) {  
        if ( a < 1 ) throw INVALID_AGE ;  
        age = a ;  
    }  
};  
int main() {  
    Age a1(10) ;  
    Age a2(-10) ; // INVALID_AGE 예외 발생  
}
```

Q & A
