

제6장

구조체와 클래스

학습 목표

- 구조체
 - 구조체형
 - 함수 인자로서 구조체
 - 구조체 초기화
- 클래스
 - 정의, 멤버 함수
 - Public과 private 멤버
 - Accessor와 mutator 함수
 - 구조체 vs. 클래스

구조체

- 두 번째 종합적인 데이터 형: 구조체
 - 모든 member가 public인 class로 이해하면 됨
- 다시 언급: 종합적이다라는 것은 그룹화하다라는 의미
 - 배열: 같은 자료형의 값의 집합
 - 구조체: 다양한 유형의 값의 집합
- 배열처럼 하나의 아이템으로 다룸
- 주요 차이점: 처음에 구조를 정의해야 함
 - 변수 선언을 우선해야 함

구조체 형

- (일반적으로) 전역적으로 구조를 정의
- 메모리에 할당되지 않음
 - 단지 정의된 구조처럼 보이긴 위한 공간 확보자
- 정의:

```
struct CDAccountV1  ← 새로운 구조체 형의 이름
{
    double balance;      ← 멤버 이름
    double interestRate;
    int term;
};
```

구조체 변수 선언과 접근

- 구조체를 정의하고 나면, 새로운 형의 변수를 선언:

```
CDAccountV1 account;
```

일반 자료형 선언과 같음

- 이제 변수 *account*는 CDAccountV1형
- 멤버 변수를 포함
 - 구조의 각각의 부분들
- 멤버에 접근하기 위한 도트(.) 연산자
 - `account.balance / account.interestRate / account.term`
- 멤버 변수
 - 구조체 변수의 부분
 - 서로 다른 구조체는 같은 멤버 변수 이름을 사용가능
 - 충돌나지 않는다.

디스플레이 6.1 구조체 정의 (1 of 3)

디스플레이 6.1 구조체 정의

```
1 //CDAccountV1 구조체형을 보여 주는 프로그램
2 #include <iostream>
3 using namespace std;

4 //예금의 은행 증명서에 대한 구조체:
5 struct CDAccountV1
6 {
7     double balance;
8     double interestRate;
9     int term; //만기일까지의 개월 수
10 };

11 void getData(CDAccountV1& theAccount);
12 //사후조건: theAccount.balance, theAccount.interestRate 그리고
13 //theAccount.term의 값은 사용자가 키보드로 입력한다.
```

이 구조체의 향상된 버전은
이 장 뒷부분에 제시될 것이다.

디스플레이 6.1 구조체 정의 (2 of 3)

```
14  int main( )
15  {
16      CDAccountV1 account;
17      getData(account);

18      double rateFraction, interest;
19      rateFraction = account.interestRate/100.0;
20      interest = account.balance*(rateFraction*(account.term/12.0));
21      account.balance = account.balance + interest;

22      cout.setf(ios::fixed);
23      cout.setf(ios::showpoint);
24      cout.precision(2);
25      cout << "When your CD matures in "
26           << account.term << " months,\n"
27           << "it will have a balance of $"
28           << account.balance << endl;

29      return 0;
30  }
```

디스플레이 6.1 구조체 정의 (3 of 3)

```
31  //iostream 사용:
32  void getData(CDAccountV1& theAccount)
33  {
34      cout << "Enter account balance: $";
35      cin >> theAccount.balance;
36      cout << "Enter account interest rate: ";
37      cin >> theAccount.interestRate;
38      cout << "Enter the number of months until maturity: ";
39      cin >> theAccount.term;
40  }
```

Sample Dialogue

```
Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity: 6
When your CD matures in 6 months,
it will have a balance of $105.00
```


구조체 함정

- 구조체 정의후에 세미콜론
 - 세미콜론(;)이 존재해야함:

```
struct WeatherData
{
    double temperature;
    double windVelocity;
}; ← 세미콜론 요구됨!
```

- 이 위치에 구조체 변수를 선언할 수 있음

```
struct WeatherData
{
    double temperature;
    double windVelocity;
} wData1, wData2 ;
```

구조체 할당과 초기화

- 2개의 구조체 변수 선언: `CropYield apples, oranges;`
 - 두 개 모두 CropYield 구조체 형 변수
 - 단순 할당은 적합:
`apples = oranges;`
 - 단순히 apples의 각 멤버 변수가 oranges의 멤버 변수에 복사됨
- 선언할 때 초기화 가능
 - 예제:

```
struct Date
{
    int month;
    int day;
    int year;
};
Date dueDate = {12, 31, 2003};
```
 - 위 선언은 3개의 멤버 변수에 데이터를 초기화

함수 인자로서의 구조체

- 기본 자료형과 같이 전달됨
 - 값에 의한 전달, 참조에 의한 전달, 또는 2가지와 조합해서
- 또한 함수에 의하여 리턴 될 수 있음
 - 리턴 유형은 구조체 형
 - 함수 정의에서 리턴 구문은 호출자에게 구조체 변수를 되돌려 줌

클래스

- 구조체와 유사
 - 멤버 함수 추가
 - 멤버 데이터만이 아님
- 객체 지향 프로그래밍의 진수
 - 객체에 초점
 - 객체 : 데이터와 기능을 포함
 - C++에서 , 클래스 형의 변수를 객체라 함

클래스 정의

- 구조체와 유사하게 정의됨
- 예제:

```
class DayOfYear  ← 새로운 클래스 형 이름
{
public:
    void output();      ← 멤버 함수!
    int month;
    int day;
};
```

- 멤버 함수의 원형만 작성하는 것에 주목
 - 함수의 구현은 어디에서든지

객체 선언 및 멤버 접근

- 모든 변수를 선언하는 것과 같음
 - 사전 정의 형, 구조체 형
- 예제: `DayOfYear today, birthday;`
 - DayOfYear 클래스 형의 2개의 객체 선언
- 객체는 다음을 포함:
 - 데이터: month, day 멤버
 - 기능(멤버 함수): output()
- 멤버는 구조체와 같이 접근됨
- 예제: `today.month, today.day`
 - 그리고 멤버 함수의 접근:
`today.output();` ← 멤버 함수 호출

클래스 멤버 함수

- 반드시 클래스 멤버 함수는 정의 또는 구현해야 함
- 다른 함수 정의와 똑같이

- main() 함수 정의 이후에 가능
- 반드시 class를 명시해야 함:

```
void DayOfYear::output()
```

```
{...}
```

- ::는 영역 지정 연산자
- 컴파일러에게 멤버 함수가 어떤 클래스의 멤버인지를 알려주는 역할
- 영역 지정 연산자 앞에 오는 클래스 이름은 형 제한자라 함

디스플레이 6.3 멤버 함수를 가지는 클래스 (1of3)

디스플레이 6.3 멤버 함수를 가지는 클래스

```
1 //클래스의 단순한 예를 보여 주는 프로그램.
2 //DayOfYear 클래스의 향상된 버전은 디스플레이 6.4에 주어진다.
3 #include <iostream>
4 using namespace std;

5 class DayOfYear
6 {
7 public:
8     void output ( );
9     int month;
10    int day;
11 };

12 int main ( )
13 {
14     DayOfYear today, birthday;
15     cout << "Enter today's date:\n";
16     cout << "Enter month as a number: ";
17     cin >> today.month;
18     cout << "Enter the day of the month: ";
19     cin >> today.day;
20     cout << "Enter your birthday:\n";
21     cout << "Enter month as a number: ";
22     cin >> birthday.month;
23     cout << "Enter the day of the month: ";
24     cin >> birthday.day;
```

이 예제에서처럼, 일반적으로 멤버 변수는 public이 아니고 private이다. 이것에 대한 논의는 이 장 뒷부분에서 다룬다.

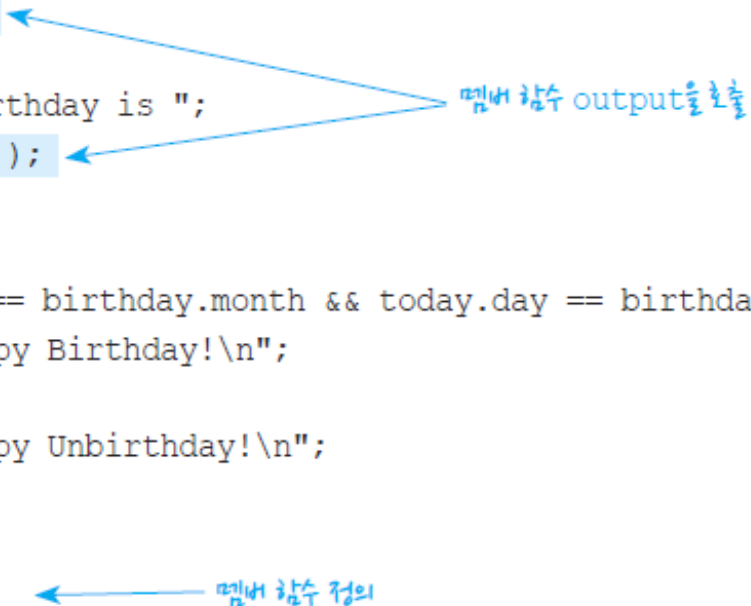
← 멤버 함수 선언

디스플레이 6.3 멤버 함수를 가지는 클래스 (2of3)

```
25     cout << "Today's date is ";
26     today.output( );
27     cout << endl;
28     cout << "Your birthday is ";
29     birthday.output( );
30     cout << endl;

31     if (today.month == birthday.month && today.day == birthday.day)
32         cout << "Happy Birthday!\n";
33     else
34         cout << "Happy Unbirthday!\n";
35     return 0;
36 }

37 //iostream 사용:
38 void DayOfYear::output( )
39 {
40     switch (month)
41     {
42     case 1:
43         cout << "January "; break;
44     case 2:
45         cout << "February "; break;
```



디스플레이 6.3 멤버 함수를 가지는 클래스 (3of3)

디스플레이 6.3 (계속)

Sample Dialogue

```
Enter today's date:  
Enter month as a number: 10  
Enter the day of the month: 15  
Enter your birthday:  
Enter month as a number: 2  
Enter the day of the month: 21  
Today's date is October 15  
Your birthday is February 21  
Happy Unbirthday!
```

```
46         case 3:  
47             cout << "March "; break;  
48         case 4:  
49             cout << "April "; break;  
50         case 5:  
51             cout << "May "; break;  
52         case 6:  
53             cout << "June "; break;  
54         case 7:  
55             cout << "July "; break;  
56         case 8:  
57             cout << "August "; break;  
58         case 9:  
59             cout << "September "; break;  
60         case 10:  
61             cout << "October "; break;  
62         case 11:  
63             cout << "November "; break;  
64         case 12:  
65             cout << "December "; break;  
66         default :  
67             cout << "Error in DayOfYear::output.";  
68     }  
69  
70     cout << day;  
71 }
```

클래스 멤버 함수 정의

- 다음 예제에서 output() 멤버 함수 정의에서
- 클래스의 멤버 데이터를 참조
 - 제한자가 없다
- 함수는 클래스의 모든 객체가 사용
 - 호출될 때 해당 객체의 데이터를 참조할 것이다.
 - 예제: `today.output();`
 - today 객체의 데이터를 출력

도트와 영역 지정 연산자

- 어떠한 클래스에 속하는지 결정하기 위해 사용
- 도트 연산자: `today.output();`
 - 특별한 객체의 멤버를 명시
- 영역 지정 연산자: `void DayOfYear::output() { }`
 - 함수 정의를 할 때 특정 클래스를 명시

클래스의 위치

- 클래스는 완전한 유형!
 - `int`, `double` 등과 같은 자료형과 같음
- 클래스 형의 변수를 가질 수 있음
 - 객체라 함
- 클래스 형의 매개변수를 가질 수 있음
 - 값에 의한 전달
 - 참조에 의한 전달
- 다른 자료형과 똑같이 클래스 형 사용 가능!

캡슐화

- 어떠한 자료형이라도 다음을 포함
 - 데이터(데이터의 범위)
 - 기능(데이터에 의해 수행될 수 있는)

- 예제:

int 형은 다음을 포함

- 데이터: -2147483648 to 2147483647
(32 bit에서 int)
- 기능 : +, -, *, /, %, 논리, 등.

- 캡슐화: 하나로 데이터와 기능을 함께 보낸다는 의미
 - 클래스에서는 데이터와 데이터 사용 기능을 명세화 해야
 - 클래스 선언 → 객체 생성
 - 객체는 다음을 캡슐화
 - 데이터 값들
 - 데이터를 통한 기능(멤버 함수)

추상 데이터형

- "Abstract"
 - 프로그래머는 자세한 부분을 모른다는 의미
- "ADT" 약어
 - 다양한 값들의 집합과 이러한 값에 따라 사전에 정의된 일련의 기본 연산으로 구성
- ADT은 종종 언어에 독립적
 - C++에서 클래스를 통해 ADT를 구현
 - C++ 클래스는 ADT를 정의한다
 - 마찬가지로 다른 언어들도 ADT를 구현함

OOP의 원칙

- 정보 감추기
 - 클래스 사용자는 작업 방식을 상세히 알 필요가 없음
- 데이터 추상화
 - ADT/클래스를 통해 데이터가 어떻게 조작되는지 사용자에게 숨김
- 캡슐화: 데이터와 기능을 함께 보내되, 자세한 부분은 숨김

Public과 Private 멤버

- 클래스 정의에서 데이터는 항상 private로 설계하라!
 - OOP 원칙 지키기
 - 사용자로부터 데이터 숨김
 - 기능을 통해 오직 접근을 허락
 - 멤버 함수
- public 아이템(보통 멤버 함수)은 사용자 접근가능
 - 앞선 예제 수정:
- 데이터는 private
- 객체가 직접 접근 불가능

```
class DayOfYear
{
public:
    void input();
    void output();
private:
    int month;
    int day;
};
```

Public과 Private 예제 2

- 주어진 앞선 예제에서
- 객체 선언: `DayOfYear today;`
- 객체 `today`는 오직 `public` 멤버만 접근 가능
 - `cin >> today.month; // 허용 안됨!`
`cout << today.day; // 허용 안됨!`
 - `public` 기능을 호출하는 것으로 대신해야 함:
 - `today.input();`
 - `today.output();`

Public과 Private 방식

- public & private를 혼합 및 조합 가능
- 일반적으로 public은 처음에 위치
 - 클래스를 이용하는 프로그래머가 사용되는 위치의 관점을 쉽게 하기 위함
 - Private 데이터는 숨겨지기 때문에 사용자와는 무관함
- 클래스 정의 외부에서 private 데이터를 변경(접근도)할 수 없음

Accessor와 Mutator 함수

- 객체는 데이터를 가지고 무엇인가를 할 필요가 있음
- Accessor 멤버 함수
 - 객체가 데이터를 읽을 수 있도록 허락
 - Get 멤버 함수라고도 함
 - 단순히 멤버 데이터를 검색
- Mutator 멤버 함수
 - 객체가 데이터를 변경하도록 허락
 - 응용을 기반으로 조작됨

인터페이스와 구현 분리

- 클래스 사용자는 클래스가 어떻게 구현되었는지 자세한 사항을 볼 필요가 없음
 - OOP의 원칙 → 캡슐화
- 사용자는 오직 규칙만 필요
 - 클래스에 대한 인터페이스라 함
 - C++에서 → public 멤버 함수와 관련된 주석
- 클래스의 구현 감추기
 - 멤버 함수 정의가 어디에 있는지
 - 사용자는 멤버 함수를 볼 필요가 없음

구조체 vs. 클래스

- 구조체
 - 일반적으로 모든 멤버가 public
 - 멤버 함수가 없음
- 클래스
 - 일반적으로 모든 데이터 멤버가 private
 - 인터페이스 멤버 함수는 public
- 기술적으로, 똑같다
 - 지각적으로, 매우 다른 매커니즘

객체 생각하기

- 프로그래밍 변화에 대한 초점
 - 이전 → 알고리즘 중심
 - OOP → 데이터
- 알고리즘은 여전히 존재
 - 알고리즘은 단순히 알고리즘 데이터에 초점을 맞춤
 - 데이터에 맞게끔 만드는 것
- 소프트웨어 솔루션 설계
 - 다양한 객체를 정의하고 그들이 어떻게 상호작용하는지에 대해 초점을 맞춤

요약

- 구조체는 서로 다른 자료형의 집합이다
- 클래스는 하나의 단위로 데이터와 함수를 조합하여 사용한다 -> 객체
- 멤버 변수와 멤버 함수
 - public → 클래스 외부에서 접근 가능
 - private → 멤버 함수의 정의에서만 접근 가능
- 클래스와 구조체 형은 함수에 형식 매개변수가 될 수 있음
- C++ 클래스 정의
 - 2개의 핵심 부분으로 분리해야 함
 - 인터페이스: 사용자가 필요로 하는 것
 - 구현: 클래스가 어떻게 동작하는지에 대한 세부사항