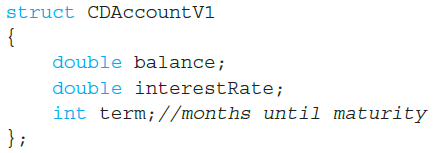
6장 구조체와 클래스

6.1 구조체

* 선언



- 구조체(structure): 서로 다른 자료형의 변수들을 한곳에 모아 놓은 사용자 자료형이다.

- 멤버 변수(데이터): 구조체 속에 있는 변수들

- 구조체 이름 = 구조체 태그

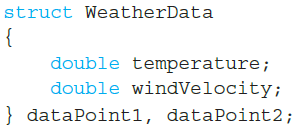
- 구조체 이름은 보통 대문자이며 모든 함수의 외부(전역)에 정의한다.

* 구조체 변수(객체)

|  |
| --- |
| CDAccountV1 account, loan; |

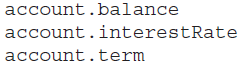
- 구조체로 선언된 변수를 구조체 변수라고 하며, 객체라고도 한다.

- 구조체 선언과 동시에 구조체 변수를 선언할 수 있다.



도트 연산자

- 구조체 변수의 멤버 변수를 접근하기 위해 사용한다.



할당 연산자

|  |
| --- |
| account = loan; |

- 하나의 구조체 변수는 **같은 형의 또 다른 구조체 변수**와 할당 연산자를 사용하면, 각 멤버 변수의 값이 일시에 처리된다.

|  |
| --- |
| account.balance = loan.balance;  account.interestRate = loan.interestRate;  account.term = loan.term; |

예제: 구조체, 할당, 도트 연산자

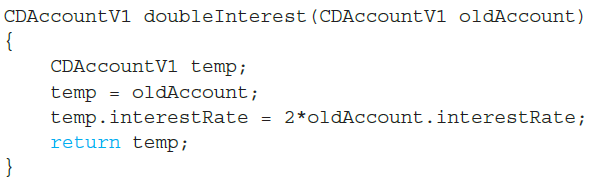
ex06\_01.cpp

|  |
| --- |
| #include<iostream> //구조체, 할당, 도트 연산자  using namespace std;  struct Accnt {  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  }; //p1, p2;  int main()  {  Accnt p1, p2; // struct Accnt p1, p2;  // 입력  cout << "ID = ? ";  cin >> p1.number;  cout << "Name = ? ";  cin >> p1.name;  cout << "Rest = ? ";  cin >> p1.value;  p2 = p1;  // 출력  cout << "ID = " << p2.number << endl;  cout << "Name = " << p2.name << endl;  cout << "Rest = " << p2.value << endl;  return 0;  } |

* 함수와 구조체

- 구조체는 call-by-vale과 call-by-reference(pointer)에 모두 사용될 수 있다.

- 구조체는 함수의 반환(return) 형에도 사용될 수 있다.



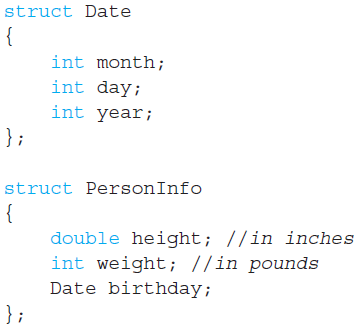
예제: 함수 call-by-value, call-by-reference, 함수 리턴값

ex06\_02.cpp

|  |
| --- |
| #include<iostream> // 함수 매개변수, 함수 리턴값, 함수 레퍼런스  using namespace std;  struct Accnt {  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  };  void input\_Accnt(Accnt& p);  // Accnt input\_Accnt();  void output\_Accnt(Accnt p);  int main()  {  Accnt p1, p2;  // 입력  // p1 = input\_Accnt();  input\_Accnt(p1);  p2 = p1;  // 출력  output\_Accnt(p2); // call-by-value  return 0;  }  //Accnt input\_Accnt()  void input\_Accnt(Accnt& p)  {  cout << "ID = ? ";  cin >> p.number;  cout << "Name = ? ";  cin >> p.name;  cout << "Rest = ? ";  cin >> p.value;  //return p;  }  void output\_Accnt(Accnt p)  {  cout << "ID = " << p.number << endl;  cout << "Name = " << p.name << endl;  cout << "Rest = " << p.value << endl;  } |

* 계층적 구조체와 초기화

- 멤버 변수로 다른 구조체를 포함할 수 있다.



- 구조체는 선언과 동시에 초기화할 수 있다. (배열의 초기화와 유사)



* C++ 구조체

- C의 구조체는 멤버 변수만 있지만, C++의 구조체는 멤버 함수를 가질 수 있다.

- 멤버 함수와 멤버 변수는 디폴트로 public이어서 다른 함수에서 접근이 가능하다. 반면에 클래스는 기본적으로 private이다.

예제: 구조체에 입/출력 함수 추가

ex06\_03.cpp

|  |
| --- |
| #include<iostream> // 구조체+입력 / 출력 함수 추가  using namespace std;  struct Accnt {  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  void input\_Accnt()  {  cout << "ID = ? ";  cin >> number;  cout << "Name = ? ";  cin >> name;  cout << "Rest = ? ";  cin >> value;  }  void output\_Accnt()  {  cout << "ID = " << number << endl;  cout << "Name = " << name << endl;  cout << "Rest = " << value << endl;  }  };  int main()  {  Accnt p1, p2;  // 입력  p1.input\_Accnt();  p2.input\_Accnt();  // 출력  p1.output\_Accnt();  p2.output\_Accnt();  return 0;  } |

예제: 구조체에 입/출력 함수, 증가/감소 함수 추가

ex06\_04.cpp

|  |
| --- |
| #include<iostream> // 구조체+입력 / 출력 함수 추가, 증가/감소 추가  using namespace std;  struct Accnt {  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  void input\_Accnt()  {  cout << "ID = ? ";  cin >> number;  cout << "Name = ? ";  cin >> name;  cout << "Rest = ? ";  cin >> value;  }  void up100\_value() {  value += 100;  }  void down100\_value() {  value -= 100;  }  void output\_Accnt()  {  cout << "ID = " << number << endl;  cout << "Name = " << name << endl;  cout << "Rest = " << value << endl;  }    };  int main()  {  Accnt p1, p2;  // 입력  p1.input\_Accnt();  p2.input\_Accnt();  // up, down 조정  p1.up100\_value();  p2.down100\_value();  // 출력  p1.output\_Accnt();  p2.output\_Accnt();  return 0;  } |

6.2 클래스

구조체와 클래스의 차이

- 구조체 = 멤버 변수 + 멤버 함수 [public]

- 클래스 = 멤버 변수 + 멤버 함수 [private]

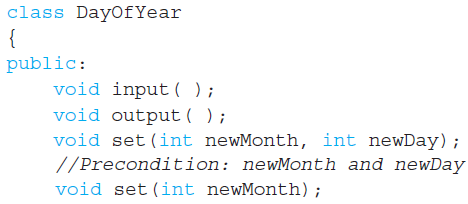
* 클래스 정의

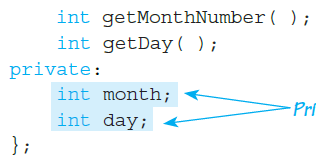
- 클래스 = 멤버 변수 + 멤버 함수의 구조체이다. 단지, 접근 제한자(private, public)가 디폴트로 private이다.

* + - 접근 제한자가 없으면 private이다.

- 따라서, C++에서는 구조체보다는 클래스가 더 많이 사용된다.

* + - private: 현재 **클래스 내의** **멤버 함수**만 접근이 가능하고, 외부에서 접근이 불가능하다.
    - public: 현재 클래스 내뿐만 아니라, 외부에서도 접근이 가능하다.
    - 캡슐화(추상화) 원칙: 멤버 변수는 private으로, 멤버 함수는 public로 한다.





- 객체(object): 클래스형의 변수를 의미한다. 객체는 멤버 변수와 멤버 함수 둘 다 가진다.



- 클래스 외부에서 접근 가능 여부

|  |
| --- |
| today.month = 12; //불가능  cin << today.day; //불가능  today.input(); //가능  birthday.set(3, 21); //가능  birthday.set(12); //가능 – 함수 오버로딩 |

예제: 클래스화, public, private

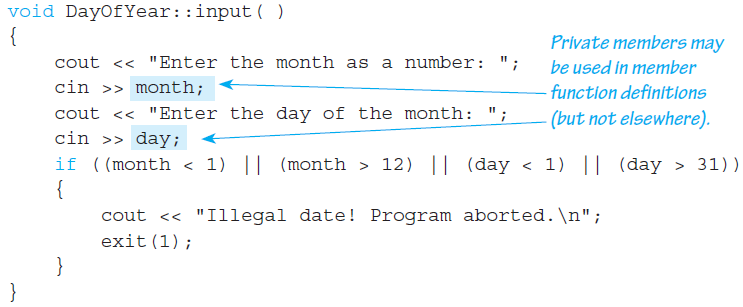
ex06\_05.cpp

|  |
| --- |
| #include<iostream> // 클래스화, public, private  using namespace std;  class Accnt {  private:  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  public:  void input\_Accnt()  {  cout << "ID = ? ";  cin >> number;  cout << "Name = ? ";  cin >> name;  cout << "Rest = ? ";  cin >> value;  }  void up100\_value() {  value += 100;  }  void down100\_value() {  value -= 100;  }  void multi\_value(double x) {  value = x \* value;  }  void output\_Accnt()  {  cout << "ID = " << number << endl;  cout << "Name = " << name << endl;  cout << "Rest = " << value << endl;  }  };  int main()  {  Accnt p1, p2;  // 입력  p1.input\_Accnt();  p2.input\_Accnt();  // up, down 조정  p1.up100\_value();  p2.down100\_value();  // multiply  double a = 3;  p1.multi\_value(a);  p2.multi\_value(2 \* a);  // 출력  p1.output\_Accnt();  p2.output\_Accnt();  return 0;  } |

* 멤버 함수의 정의

- 멤버 함수는 클래스 내에서 정의할 수도 있고, 영역지정 연산자(::)를 이용하여 클래스 외부에 정의할 수 있다.

- 보통 멤버 함수는 클래스 내에서 원형만 선언하고, 함수 정의는 클래스 외부에 몸체를 정의하는 것이 일반적인데 이것은 클래스의 헤더와 정의를 분리하기 위해서이다.



- 멤버 함수는 private 멤버 변수일지라도, 같은 클래스내의 private 멤버 변수는 접근할 수 있다.

영역지정(::) 연산자와 도트(.) 연산자의 구분

- 영역지정(::) 연산자: 클래스 이름과 같이 사용된다. 즉, 클래스 외부에 멤버 함수를 정의할 때, 함수가 어느 클래스의 멤버 함수인지 알려준다.

* + - 클래스\_이름::함수\_이름(매개 변수)
    - 멤버 함수 정의 시에는 도트(.) 연산자를 사용할 수 없다.

- 도트(.) 연산자: 객체(클래스 변수)와 함께 사용된다. 어떤 객체의 멤버 함수(변수)를 호출할 때 사용한다.

* + - 객체\_이름.멤버\_함수(매개 변수);



…

}



예제: 멤버 함수 분리

ex06\_06.cpp

|  |
| --- |
| #include<iostream> // 멤버함수 분리  using namespace std;  class Accnt {  private:  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  public:  void input\_Accnt();  void up100\_value();  void down100\_value();  void multi\_value(double x);  void output\_Accnt();  };  int main()  {  Accnt p1, p2;  // 입력  p1.input\_Accnt();  p2.input\_Accnt();  // up, down 조정  p1.up100\_value();  p2.down100\_value();  // multiply  double a = 3;  p1.multi\_value(a);  p2.multi\_value(2 \* a);  // 출력  p1.output\_Accnt();  p2.output\_Accnt();  return 0;  }  void Accnt::input\_Accnt()  {  cout << "ID = ? ";  cin >> number;  cout << "Name = ? ";  cin >> name;  cout << "Rest = ? ";  cin >> value;  }  void Accnt::up100\_value() {  value += 100;  }  void Accnt::down100\_value() {  value -= 100;  }  void Accnt::multi\_value(double x) {  value = x \* value;  }  void Accnt::output\_Accnt()  {  cout << "ID = " << number << endl;  cout << "Name = " << name << endl;  cout << "Rest = " << value << endl;  } |

* Accessor 함수와 Mutator 함수 (입출력 함수)

- Accessor 함수: 객체내의 자료를 읽을 수 있도록 하는 함수. 보통 get으로 시작한다.

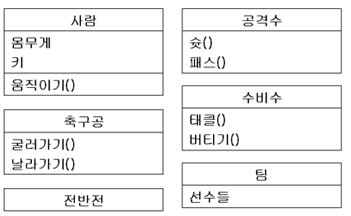
- Mutator 함수: 객체내의 자료를 바꾸는 함수. 보통 set으로 시작한다.

* 객체지향 프로그래밍과 캡슐화

객체지향 프로그래밍

- 객체란? 현실 세계에 존재하는 개념들 중에서 소프트웨어 개발 대상이 되는 것으로 객체의 속성(멤버 변수)과 객체의 행동 방식(멤버 함수)으로 구성되어 있다.

- 객체지향 프로그램: 객체가 가진 속성(데이터)보다 객체 사이의 상호 동작에 주안점을 두어 프로그램 하는 방식이다.



캡슐화

- 캡슐화는 클래스의 정의에 있어서 멤버 변수 및 함수의 구현이 어떻게 이루어져 있는지 클래스 이용자에게 숨기거나 최소한으로 이루어져야 함을 의미한다.

- 캡슐화 원칙에 의해 프로그래머는 클래스의 구현 부분은 신경 쓰지 않고, 클래스의 인터페이스 부분만 참고하여 프로그램 할 수 있어야 한다.

- 따라서 개발자는 클래스의 정의(인터페이스) 부분을 다른 사용자를 위해서 잘 문서화 해야 한다. 이와 같이 인터페이스와 구현이 분리된 클래스를 캡슐화된 클래스라 한다.

- 이와 같이 캡슐화된 클래스는 실제 세계의 객체를 잘 표현할 수 있으며 객체 사이의 상호 작용을 프로그램으로 표현할 수 있다.

예제: 클래스 인터페이스와 구현 분리

- 클래스의 헤더(Accnt.h)와 구현(Accnt.cpp)는 사용하고자 하는 함수(ex06\_07.cpp)와 같은 프로젝트에 있어야 한다.

ex06\_07.cpp

|  |
| --- |
| #include "Accnt.h"  int main()  {  Accnt p1, p2;  // 입력  p1.input\_Accnt();  p2.input\_Accnt();  // up, down 조정  p1.up100\_value();  p2.down100\_value();  // multiply  double a = 3;  p1.multi\_value(a);  p2.multi\_value(2 \* a);  // 출력  p1.output\_Accnt();  p2.output\_Accnt();  return 0;  } |

Accnt.h

|  |
| --- |
| #include<iostream> // 멤버함수 분리  using namespace std;  class Accnt {  private:  int number; // 계좌번호  char name[10]; // 예금주  double value; // 잔액  public:  void input\_Accnt();  void up100\_value();  void down100\_value();  void multi\_value(double x);  void output\_Accnt();  }; |

Accnt.cpp

|  |
| --- |
| #include <iostream>  #include "Accnt.h"  void Accnt::input\_Accnt()  {  cout << "ID = ? ";  cin >> number;  cout << "Name = ? ";  cin >> name;  cout << "Rest = ? ";  cin >> value;  }  void Accnt::up100\_value() {  value += 100;  }  void Accnt::down100\_value() {  value -= 100;  }  void Accnt::multi\_value(double x) {  value = x \* value;  }  void Accnt::output\_Accnt()  {  cout << "ID = " << number << endl;  cout << "Name = " << name << endl;  cout << "Rest = " << value << endl;  } |