7장 생성자와 툴

7.0 복습

예제: 클래스

ex07\_01.cpp

|  |
| --- |
| // 본 예제는 스마트학습관리시스템에서 다운 받을 수 있음  #include <iostream> // DayOfYear 클래스 선언  #include <string>  using namespace std;  class DayOfYear  {  public:  void input();  void output();  string getMonth();  int getDay();  void setMonthDay(string m, int d);  private:  string month;  int day;  };  int main()  {  DayOfYear date1, date2, date3;    // 입력  date1.input();    // 변경  date1.setMonthDay("Jan", 3);    // 출력  date1.output();    return 0;  }  void DayOfYear::input()  {  cout << "Month & day: ";  cin >> month >> day;  }  void DayOfYear::output()  {  cout << "Month = " << month << endl;  cout << "Day = " << day << endl;  }  string DayOfYear::getMonth()  {  return month;  }  int DayOfYear::getDay()  {  return day;  }  void DayOfYear::setMonthDay(string m, int d)  {  month = m;  day = d;  } |

7.1 생성자

- 클래스 객체를 초기화하는데 사용되는 멤버 함수이다.

- 생성자(constructor)는 클래스의 정의에 멤버 함수로 포함되며, 객체가 선언되면 자동호출 된다.

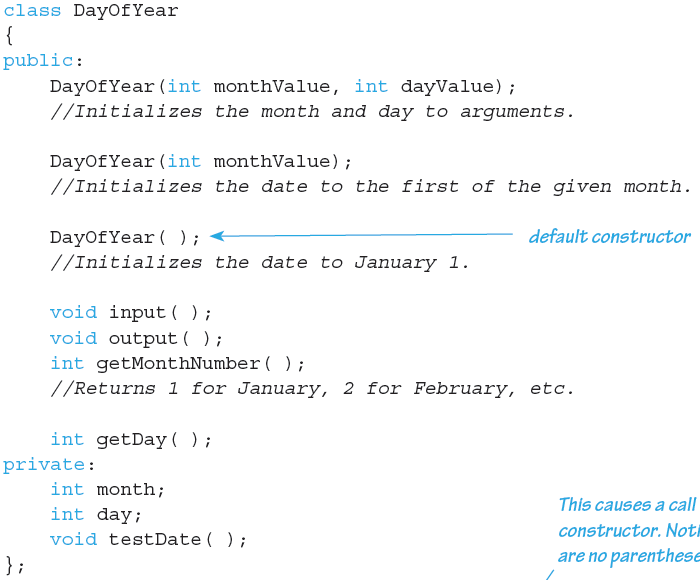
* 생성자 선언

- 생성자는 클래스와 같은 이름을 가져야 한다.

- 생성자 정의에는 어떤 값도 리턴할 수 없다. void 형도 사용해서는 안 된다.

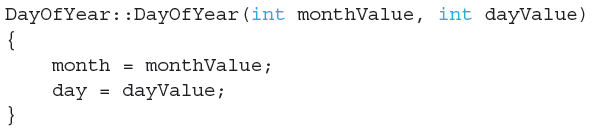
- 생성자는 public 영역에 있어야 한다.

- 생성자는 일반 멤버 함수처럼 오버로딩 될 수 있다.



* 생성자 정의

- 생성자는 다른 멤버 함수와 같은 방법으로 정의한다.

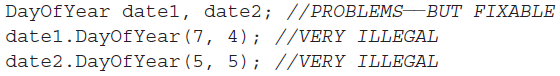


* 생성자 호출

- 객체가 선언되고 그 다음에 생성자가 호출되어 초기화 한다.



- 생성자는 일반 멤버 함수를 호출하는 것과 같은 방법으로 호출할 수 없다.



- 인자가 없는 생성자를 호출할 때는 괄호를 사용하지 않는다.

* 디폴트 생성자(default constructor)

- 인자를 취하지 않는 생성자를 의미한다.

- 클래스 정의 시, 생성자를 하나라도 포함하지 않는 경우,

* + - 컴파일러는 디폴트 생성자를 자동 생성한다.

하나라도 포함한 경우

* + - 반드시 포함시켜야 한다.



그러지 않으면 date3을 초기화 하지 못한다.

즉, 모든 생성자 선언 시, 1개의 디폴트 생성자를 포함해야 한다.

|  |
| --- |
| DayOfYear::DayOfYear()  { /\* Do nothing \*/ } |

예제: 생성자 구현

ex07\_02.cpp

|  |
| --- |
| #include <iostream> // DayOfYear 생성자  #include <string>  using namespace std;  class DayOfYear  {  public:  DayOfYear(string m, int d);  DayOfYear(string m);  DayOfYear();  void input();  void output();  string getMonth();  int getDay();  void setMonthDay(string m, int d);  private:  string month;  int day;  };  int main()  {  DayOfYear date1("May", 3), date2("Feb"), date3; // date3(); 오류  // 출력  date1.output();  date2.output();  date3.output();  return 0;  }  DayOfYear::DayOfYear(string m, int d) {  month = m;  day = d;  }  DayOfYear::DayOfYear(string m) {  month = m;  day = 1;  }  DayOfYear::DayOfYear() {  month = "Jan";  day = 1;  }  void DayOfYear::input()  {  cout << "Month & day: ";  cin >> month >> day;  }  void DayOfYear::output()  {  cout << "Month = " << month << endl;  cout << "Day = " << day << endl;  }  string DayOfYear::getMonth()  {  return month;  }  int DayOfYear::getDay()  {  return day;  }  void DayOfYear::setMonthDay(string m, int d)  {  month = m;  day = d;  } |

* 명시적 호출과 초기화 섹션

명시적 호출

- 객체는 클래스 변수의 이름이 없이 생성되어, 초기화 되고 다른 객체에 할당될 수 있다. (무명 객체)



- 명시적 호출이란 생성자가 무명의 객체를 생성하여 리턴하는 함수처럼 작동하는 것을 의미한다.

- 인자가 없는 생성자의 명시적 호출은 괄호를 반드시 사용해야 한다.

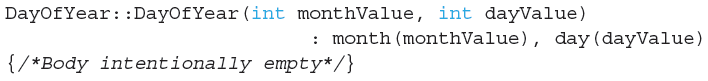


초기화 섹션(Initialization section)

- 함수 헤더와 { }사이에 위치한다.

- 콜론(:) 뒤에 멤버 변수(매개변수) 형식으로 쉼표(,)로 구분하여 구성한다.

- 『멤버 변수 = 매개 변수』인 관계에서만 사용한다.



예제: 명시적 호출과 초기화 섹션

ex07\_03.cpp

|  |
| --- |
| #include <iostream> // 명시적 호출과 초기화 섹션  #include <string>  using namespace std;  class DayOfYear  {  public:  DayOfYear(string m, int d);  DayOfYear(string m);  DayOfYear();  void input();  void output();  string getMonth();  int getDay();  void setMonthDay(string m, int d);  private:  string month;  int day;  };  int main()  {  DayOfYear date1, date2, date3;  // 명시적 호출  date1 = DayOfYear("May", 3);  date2 = DayOfYear("Feb");  date3 = DayOfYear();  // 출력  date1.output();  date2.output();  date3.output();  return 0;  }  // 초기화 섹션  DayOfYear::DayOfYear(string m, int d):month(m), day(d)  {  }  DayOfYear::DayOfYear(string m):month(m), day(1)  {  }  DayOfYear::DayOfYear():month("Jan"), day(1)  {  }  void DayOfYear::input()  {  cout << "Month & day: ";  cin >> month >> day;  }  void DayOfYear::output()  {  cout << "Month = " << month << endl;  cout << "Day = " << day << endl;  }  string DayOfYear::getMonth()  {  return month;  }  int DayOfYear::getDay()  {  return day;  }  void DayOfYear::setMonthDay(string m, int d)  {  month = m;  day = d;  } |

* 클래스형 멤버 변수

- 클래스의 멤버 변수가 또 다른 클래스일 수 있다.

- 클래스가 상속관계에 있을 때, 하위 클래스에서 상위 클래스를 초기화 할 때 초기화 섹션을 이용한다.

예제: 클래스의 상속과 초기화 섹션

ex07\_04.cpp

|  |
| --- |
| #include <iostream> // 클래스의 상속과 초기화 섹션  using namespace std;  class DayOfYear  {  public:  DayOfYear(string m, int d);  DayOfYear(string m);  DayOfYear();  void input();  void output();  string getMonth();  int getDay();  void setMonthDay(string m, int d);  private:  string month;  int day;  };  class Holiday {  public:  Holiday(string m, int d, bool h);  Holiday();  void output();  private:  DayOfYear date;  bool tf;  };  int main()  {  Holiday children("May",5,true), today("April", 29, false), anydate;  // 출력  children.output();  today.output();  anydate.output();  return 0;  }  //Holiday 생성자  Holiday::Holiday(string m, int d, bool h):date(m, d),tf(h)  {  }  Holiday::Holiday() : date(), tf(true)  {  }  void Holiday::output()  {  date.output();  cout << "Holiday = " << tf << endl;  }  // DayOfYear 생성자  DayOfYear::DayOfYear(string m, int d):month(m), day(d)  {  }  DayOfYear::DayOfYear(string m):month(m), day(1)  {  }  DayOfYear::DayOfYear():month("Jan"), day(1)  {  }  void DayOfYear::input()  {  cout << "Month & day: ";  cin >> month >> day;  }  void DayOfYear::output()  {  cout << "Month = " << month << endl;  cout << "Day = " << day << endl;  }  string DayOfYear::getMonth()  {  return month;  }  int DayOfYear::getDay()  {  return day;  }  void DayOfYear::setMonthDay(string m, int d)  {  month = m;  day = d;  } |

7.2 그외의 툴

* 클래스에 적용되는 const 수정자

1. 멤버 변수에 const를 붙이는 경우

- 값이 변하지 않는 상수 멤버를 만든다.

- 상수 멤버는 생성자에서 초기화 목록으로 초기화 한다.

|  |
| --- |
| class Car  {  public:  Car(int sl, int s=0, string c=”white”): serial(sl), speed(s), color(c)  {  }  private:  **const** int serial;  int speed;  string color;  }  …  Car c1(6091, 200, “blue”), c2(4898); |

1. 멤버 함수에 const를 붙이는 경우 (상수 멤버 함수)

- const가 붙은 멤버 함수는 멤버 변수를 변경할 수 없다. 즉, const 멤버 함수는 멤버 변수의 값을 읽을 수는 있으나 변경할 수는 없다.

- 함수의 선언과 정의에서 가장 오른쪽 끝에 const를 추가한다.

|  |
| --- |
| class Car  {  public:  Car(int sl, int s=0, string c=”white”): serial(sl), speed(s), color(c)  {  }  **void displySpeed() const // speed 값을 변경할 수 없다.**  **{**  **cout << “Car Speed = ” << speed << endl;**  **}**  private:  const int serial;  int speed;  string color;  } |

- 상수 멤버 함수 안에서 상수화되지 않은 함수를 호출하면 오류가 발생한다 (1번). 따라서 멤버 변수를 변경하지 않는다고 할지라도 가능성이 있으므로 호출하는 함수도 const를 붙인다. (2번)

|  |
| --- |
| class Car  {  public:  …  **// int getSpeed() // (1번) 오류**  **int getSpeed() const // (2번) 가능**  {  return speed;  }  void displySpeed() **const**  {  cout << “Car Speed = ” << **getSpeed()** << endl;  }  private:  const int serial;  int speed;  string color;  } |

1. 객체에 const를 붙이는 경우

- 이 객체의 멤버 변수는 값을 변경할 수 없으며, 상수 멤버 함수만이 객체의 멤버 변수를 호출할 수 있다.

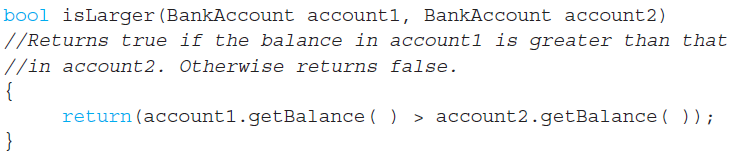
|  |
| --- |
| int main  {  const Car c3(9876, 300, “red”);  c3.setSpeed(); // 오류 - setspeed()는 speed 값을 변경한다는 가정하에  c3.getSpeed(); // 가능 **int getSpeed() const**  …  } |

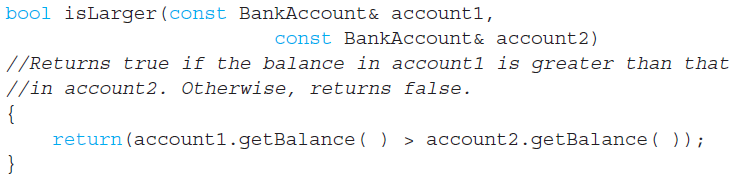
1. 매개변수에 const를 붙이는 겨우

매개변수가 클래스인 경우 const 매개변수를 사용하자.

- 클래스는 함수의 매개변수로 call-by-value나 call-by-reference에 둘 다 사용될 수 있다.

* + - call-by-reference가 메모리(변수)를 더 적게 사용하기 때문에 효율적이고 일반적이다.
    - 만약, 원본(인자)의 값이 변경되지 않으면 매개변수 앞에 const 수정자를 붙인다. (call-by-value 역할)
    - const 매개변수는 자동으로 에러를 검출하는 역할을 한다.





* + - 비록 같은 기능을 하지만, 아래의 코드가 메모리와 속도 면에서 더 효율적이다.

일관적인 const 사용

- const 수정자는 사용하려면 항상 사용하거나, 아니라면 아예 사용하지 않아야 한다.

- 어떤 함수의 매개변수로 클래스가 사용되고 거기에 const가 붙으면 상수화되지 않은 멤버 함수는 호출할 수 없다. 즉, 호출하기 위해서는 const를 붙여야 한다.



|  |
| --- |
| void sub(Car& c1, **const Car& c2**)  {  int s1=c1.getSpeed(); // 가능  c1.setSpeed(250); // 가능  **int s2=c2.getSpeed(); // 가능**  **c2.setSpeed(150); // 오류**- setspeed()는 speed 값을 변경한다는 가정하에  } |

* 인라인(inline) 함수

- 클래스의 정의 안에 멤버 함수를 **완전하게** 정의하는 함수이다.

- 함수의 정의가 매우 짧을 때 사용한다.

- 멤버 함수 호출 시, 코드가 그대로 삽입된다.

- 클래스를 인터페이스와 정의부로 나눌 수 없다.

* 정적(static) 멤버

정적 멤버 변수(클래스 변수)

- 정적 변수(static variable)란 같은 클래스의 모든 객체가 공유(1개)하는 변수이다.

- 객체들이 서로 대화하거나 행동을 조정하기 위하여, 정적 변수를 두어 특정 멤버 함수로 공유한다.

- 같은 클래스의 객체만이 직접 접근할 수 있도록 private이 될 수 있다. 즉, 객체들의 전역 변수 역할을 한다.

- 정적 변수의 초기화는 클래스의 외부에 한번만 선언한다.

정적 멤버 함수

- 정적 멤버 함수는 클래스의 모든 객체들이 공유한다.

- 정적 멤버 함수는 클래스의 객체를 만들지 않고서도 호출될 수 있다.

- 정적 멤버 함수의 정의에서는 호출 객체에 의해 좌우되는 어떤 것도 사용할 수 없다. 즉, 정적 멤버 변수만 사용할 수 있다.

* + - 일반 멤버 함수를 호출할 수 없다.

- static 키워드는 멤버 함수의 정의에서는 사용하지 않는다.

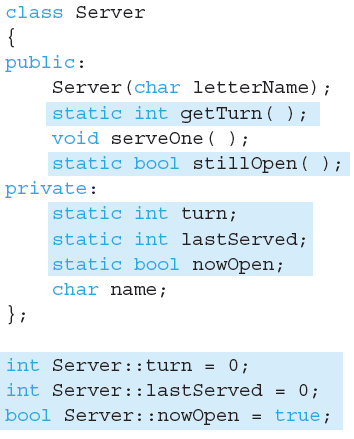
- 객체가 없으므로 클래스의 이름을 객체처럼 사용하여 호출한다.

정적 상수 (const static)

- 정적 멤버 변수를 상수화 한 것이다.

- 모든 객체들을 통틀어서 하나만 존재하고 선언과 동시에 초기화 가능하고 초기화 된 후 그 값을 변경할 수 없다.

- 정수형만 가능하다.

 교재 내용 참고

예제: static 변수, 함수, 상수

ex07\_05.cpp

|  |
| --- |
| #include <iostream> // static 변수, 함수, 상수  #include <string>  using namespace std;  class DayOfYear  {  public:  DayOfYear(string m, int d);  DayOfYear(string m);  DayOfYear();  void input();  void output();  string getMonth();  int getDay();  void setMonthDay(string m, int d);  int setfreeTime() // static 변수 조정함수 , 1-3  {  freeTime ++;  freeTime = freeTime + day; // 일반 멤버변수 호출가능  return freeTime;  }  static int addfreeTime() // static 함수, 2-1, setfreeTime와 비교  {  freeTime = freeTime + 10;  // freeTime = freeTime + day; // 일반 멤버변수 호출 불가능 2-2  return freeTime;  }  //int getmax\_day() const // 3-2  static int getmax\_day() // 3-2  {  return MAX\_DAY;  }  private:  static int freeTime; // static 변수, 1-1  const static int MAX\_DAY = 365; // static 상수, 3-1  string month;  int day;  };  int DayOfYear::freeTime = 0; // static 변수 초기화 , 1-2  int main()  {  DayOfYear date1("May", 3), date2("March"), date3;  // static 변수  cout << date1.setfreeTime() << endl; // 1- 출력  cout << date2.setfreeTime() << endl; // 1- 출력  cout << date3.setfreeTime() << endl; // 1- 출력  // static 함수  cout << date1.addfreeTime() << endl; // 2- 출력  cout << date2.addfreeTime() << endl; // 2- 출력  cout << DayOfYear::addfreeTime() << endl; // 2- 출력  // static 상수  cout << date1.getmax\_day() << endl; // 3- 출력  cout << DayOfYear::getmax\_day() << endl; // 2- 출력  return 0;  }  // 초기화 섹션  DayOfYear::DayOfYear(string m, int d):month(m), day(d)  {  }  DayOfYear::DayOfYear(string m):month(m), day(1)  {  }  DayOfYear::DayOfYear():month("Jan"), day(1)  {  }  void DayOfYear::input()  {  cout << "Month & day: ";  cin >> month >> day;  }  void DayOfYear::output()  {  cout << "Month = " << month << endl;  cout << "Day = " << day << endl;  }  string DayOfYear::getMonth()  {  return month;  }  int DayOfYear::getDay()  {  return day;  }  void DayOfYear::setMonthDay(string m, int d)  {  month = m;  day = d;  } |

7.3 벡터: 표준 템플릿 라이브러리의 미리보기

* 벡터의 기본

- 벡터는 프로그램이 실행되는 동안 길이가 커지거나 줄어들 수 있는 배열이다.

- vector<기본형>의 표현을 가지며, 표준 템플릿 라이브러이(STL)에 있는 템플릿 클래스로 형성되어 있다. #include <vector>

- 선언

|  |
| --- |
| vector<int> v1; // 정수형 빈 객체를 생성하는 디폴트 생성자 호출  vector<DayOfYear> v2(5); // DayOfYear 클래스형 객체 5개를 생성하는 생성자 호출 |

* + - 인텍스는 0부터 시작한다.
    - 인텍스로 [숫자]를 사용한다.

- 여러 가지 함수

* + - 벡터에 원소 추가

|  |
| --- |
| v1.push\_back(10); // 첫 번째 원소에 10 추가 |

* + - 벡터의 마지막 원소 제거

|  |
| --- |
| v1.pop\_back(); // 마지막 원소 제거 |

* + - 벡터의 사이즈

|  |
| --- |
| v1.size(); // unsigned int 값  v1.resize(15); //v1의 크기를 15로 변경 |

예제: vector의 사용

ex07\_06.cpp

|  |
| --- |
| #include<iostream> // vector 사용  #include<vector>  using namespace std;  int main()  {  vector<int> a;  //int k;  //cin >> k;  //a.push\_back(k);  //cin >> k;  //a.push\_back(k);  //a.push\_back(6);  //for (unsigned int i = 0; i < a.size(); i++)  // cout << a[i] << " ";  //cout << endl;  vector<int> b(5);  for (unsigned int i = 0; i < b.size(); i++)  cout << b[i] << " ";  cout << endl;  b.push\_back(11);  b.push\_back(12);  b.push\_back(13);  b[1] = 9;  for (unsigned int i = 0; i < b.size(); i++)  cout << b[i] << " ";  cout << endl;    b.pop\_back();  b.pop\_back();  b.resize(15);  for (unsigned int i = 0; i < b.size(); i++)  cout << b[i] << " ";  cout << endl;  return 0;  } |