제7장

생성자와 틀

학습 목표

- 생성자
  - 정의
  - 호출
- 틀
  - const 매개변수 수정자
  - 인라인 함수
  - 정적 멤버 데이터
- 벡터
  - 벡터 클래스 소개

생성자

- 일부 또는 모든 멤버 변수 초기화. 다른 초기화 액션도 가능
- 특별한 종류의 멤버 함수로 객체 선언시 자동 호출

- 생성자를 포함하는 클래스 정의:

```
class DayOfYear
{
public:
    DayOfYear(int monthValue, int dayValue);
                        // month와 day를 초기화하는 생성자
    void input();
    void output();
    …
private:
    int month;
    int day;
}
```

생성자에서 주목할 점

- 생성자 이름 : DayOfYear
  - 클래스 자체 이름과 같음!
- 생성자 선언은 리턴 유형이 없음
  - 심지어 void도 아님!
- public 섹션에서 생성자
  - 객체가 선언될 때 호출됨
  - 만약 private라면, 객체를 선언할 수 없다!

- Java의 this() 생성자 사용 불가

생성자 호출

- 객체 선언:

```
DayOfYear date1(7, 4), date2(5, 5);
```

- 객체가 생성되면
  - 생성자가 호출되고, 생성자의 인자로서 매개변수에 값이 전달
  - 멤버 변수 month, day는 다음과 같이 초기화:

    date1.month → 7 date2.month → 5

    date1.day → 4 date2.day → 5

- 다음을 고려하자:

```
DayOfYear date1, date2 ;
date1.DayOfYear(7, 4);      // 부적합! 컴파일 오류
date2.DayOfYear(5, 5);      // 부적합! 컴파일 오류
```

  - 겉보기에는 괜찮아 보이지만...

    생성자는 다른 멤버 함수처럼 호출할 수 없음!

생성자 코드

- 생성자 정의는 모든 다른 멤버 함수와 같음:

```
DayOfYear::DayOfYear(int monthValue, int dayValue)
{
        month = monthValue;
        day = dayValue;
}
```

  - 영역 지정 연산자(::) 앞과 뒤의 이름이 동일
  - 리턴 유형이 없다는 것에 주목

- 생성자 정의의 다른 형태

```
DayOfYear::DayOfYear(int m, int d)  : month(m), day(d)  // ←
{…}
```

  - 위의 코드와 동일한 기능. ":" 뒤의 ← 부분을 "초기화 섹션"이람 함
  - 몸통은 공백으로 남겨둠
  - 선호되는 정의 버전

생성자의 주요 역할 및 생성자 오버로드

- 데이터 초기화 뿐만 아니라,
  클래스 private 멤버 변수에 적절한 데이터가 할당되도록 보장
  - 강력한 OOP 원칙

- 다른 함수와 같이 생성자도 오버로드 가능
  - 함수의 서명은 함수의 이름과 매개변수 목록을 포함

- 생성자는 가능한 모든 매개변수 목록을 제공

디스플레이 7.1 생성자를 가지는 클래스 (1 of 3)

디스플레이 7.1 생성자를 가진 클래스

```
1   #include <iostream>
2   #include <cstdlib> //exit 때문에
3   using namespace std;
                                  이 DayOfYear 정의는 디스플레이 6.4의
4   class DayOfYear               DayOfYear 클래스를 개정한 것이다.
5   {
6   public:
7       DayOfYear(int monthValue, int dayValue);
8       //month와 day를 인자 값으로 초기화한다.

9       DayOfYear(int monthValue);
10      //날짜를 주어진 달의 첫째 날로 초기화한다.

11      DayOfYear( ); ←———————— 디폴트 생성자
12      //날짜를 1월 1일로 초기화한다.

13      void input( );
14      void output( );
15      int getMonthNumber( );
16      //1월은 1, 2월은 2 등과 같이 리턴한다.

17      int getDay( );
```

```
18  private:
19      int month;
20      int day;
21      void testDate( );
22  };
```

```
23   int main( )
24   {
25       DayOfYear date1(2, 21), date2(5), date3;
26       cout << "Initialized dates:\n";
27       date1.output( ); cout << endl;
28       date2.output( ); cout << endl;
29       date3.output( ); cout << endl;

30       date1 = DayOfYear(10, 31);
31       cout << "date1 reset to the following:\n";
32       date1.output( ); cout << endl;
33       return 0;
34   }
35
```

이로 인해 디폴트 생성자가 호출된다.
매개변수가 없는 것에 주목한다.

DayOfYear::DayOfYear
생성자에 대한 명시적 호출

Sample Dialogue

```
Initialized dates:
February 21
May 1
January 1
date1 reset to the following:
October 31
```

```
36   DayOfYear::DayOfYear(int monthValue, int dayValue)
37                : month(monthValue), day(dayValue)
38   {
39       testDate( );
40   }
41   DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
42   {
43       testDate( );
44   }

45   DayOfYear::DayOfYear( ) : month(1), day(1)
46   { /*본체는 의도적으로 비워 두었다.*/}
```

```
47   //iostream과 cstdlib 사용:
48   void DayOfYear::testDate( )
49   {
50       if ((month < 1) || (month > 12))
51       {
52           cout << "Illegal month value!\n";
53           exit(1);
54       }
55       if ((day < 1) || (day > 31))
56       {
57           cout << "Illegal day value!\n";
58           exit(1);
59       }
60   }
```

<나머지 멤버 함수의 정의는
디스틸레이 6.4에 있는 것과 똑같다.>

명시적 생성자 호출

- 생성자를 다시 호출 가능
  - 객체를 선언한 이후에. 즉, 생성자가 자동으로 호출된 다음에
  - 객체의 이름을 통해 호출 가능; 기본 멤버 함수 호출
- 멤버 변수를 설정하는 편리한 방법
- 기본 멤버 함수 호출과는 매우 다른 방법

- "익명 객체"를 리턴하는 호출과 같음

```
DayOfYear holiday(7, 4);   // 자동 생성자 호출
…
holiday = DayOfYear(5, 5);  // 명시적 생성자 호출.
                            // 새로운 익명 객체를 리턴하고 할당
```

디폴트 생성자

- 디폴트 생성자는 인자가 없는 생성자

```
DayOfYear date1;// 이 방법!   DayOfYear date1 = DayOfYeat() ; 와 동일
DayOfYear date2() ; // 이렇게 써도 될까?
```

- 자동 생성?
  - 만약 정의된 생성자가 하나도 없다면 → 그렇다
  - 생성자 하나라도 정의되었다면 → 아니오
- 만약 디폴트 생성자가 없다면:
  - 다음처럼 선언할 수 없음: MyClass myObject;

클래스 형 멤버 변수

- 클래스 멤버 변수는 어떠한 유형도 가능
  - 다른 클래스의 객체도 포함!
  - 클래스의 유형 관계
    - 강력한 OOP 원칙
- 생성자를 위한 특별한 표현이 요구됨
  - 다시 말하면, 멤버 객체의 생성자를 “다시” 호출 가능

디스플레이 7.3 클래스 멤버 변수 (1 of 4)

디스플레이 7.3 클래스 멤버 변수

```
1   #include <iostream>
2   #include<cstdlib>
3   using namespace std;

4   class DayOfYear
5   {
6   public:
7       DayOfYear(int monthValue, int dayValue);
8       DayOfYear(int monthValue);
9       DayOfYear( );
10      void input( );
11      void output( );
12      int getMonthNumber( );
13      int getDay( );
14  private:
15      int month;
16      int day;
17      void testDate( );
18  };
```

DayOfYear 클래스는 디스플레이 7.1에 있는 것과 똑같지만 여기서 논의하는 데 필요하므로 자세한 내용을 다시 나타내었다.

```
29  int main( )
30  {
31      Holiday h(2, 14, true);
32      cout << "Testing the class Holiday.\n";
33      h.output( );
34          return 0;
35  }
```

```
19  class Holiday
20  {
21  public:
22      Holiday( ); //주차 규정이 없는 1월 1일로 초기화한다.
23      Holiday(int month, int day, bool theEnforcement);
24      void output( );
25  private:
26      DayOfYear date;
27      bool parkingEnforcement; //true if enforced
28  };
```

클래스형의 멤버 변수

디스플레이 7.3 클래스 멤버 변수 (2 of 4)

```
42   void Holiday::output( )
43   {
44       date.output( );
45       cout << endl;
46       if (parkingEnforcement)
47           cout << "Parking laws will be enforced.\n";
48       else
49           cout << "Parking laws will not be enforced.\n";
50   }

51   DayOfYear::DayOfYear(int monthValue, int dayValue)
52                : month(monthValue), day(dayValue)
53   {
54       testDate( );
55   }
```

```
37   Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
38   {/*의도적으로 비워 두었다.*/}

39   Holiday::Holiday(int month, int day, bool theEnforcement)
40                       : date(month, day), parkingEnforcement(theEnforcement)
41   {/*의도적으로 비워 두었다*/}
```

DayOfYear 클래스의 생성자를 호출한다.

디스플레이 7.3 클래스 멤버 변수 (4 of 4)

```
71  //iostream을 사용한다:
72  void DayOfYear::output( )
73  {
74      switch (month)
75      {
76          case 1:
77              cout << "January "; break;
78          case 2:
79              cout << "February "; break;
80          case 3:
81              cout << "March "; break;
                .
                .
                .
82          case 11:
83              cout << "November "; break;
84          case 12:
85              cout << "December "; break;
86          default:
87              cout << "Error in DayOfYear::output.";
88      }

89      cout << day;
90  }
```

생략한 줄들은 디스플레이 6.3에 나타나 있지만 참조할 필요가 없을 정도로 명백히 알 수 있다.

Sample Dialogue

```
Testing the class Holiday.
February 14
Parking laws will be enforced.
```

```
56  //iostream과 cstdlib를 사용한다:
57  void DayOfYear::testDate( )
58  {
59      if ((month < 1) || (month > 12))
60      {
61          cout << "Illegal month value!\n";
62          exit(1);
63      }
64      if ((day < 1) || (day > 31))
65      {
66          cout << "Illegal day value!\n";
67          exit(1);
68      }
69  }
```

매개 변수 전달 방법

- 매개변수 전달의 효율성
  - 값에 의한 호출
    - 복사본 요구 → 오버헤드
  - 참조에 의한 호출
    - 실제 인자를 위한 공간 확보자
    - 대부분 효율적인 방법
  - 기본 유형에 대해서는 차이를 무시해도 됨
  - 클래스에 대해 → 명백한 이점
- 참조에 의한 호출이 바람직하다.
  - 특히, 클래스 형과 같은 큰 데이터에 대해

const 매개변수 수정자

- 큰 데이터 형 (일반적으로 클래스들)
  - 참조에 의한 전달을 사용하는 것이 바람직하다.
  - 심지어, 함수 매개변수의 값을 변경하지 않더라도
- 인자 보호하기
  - constant 매개변수 사용하기
    - 또한 constant call-by-reference 매개변수라고도 함
  - 유형 앞에 const 키워드를 사용
  - 매개변수를 "읽기 전용"으로 만듦
  - 매개변수를 수정하려 시도하면 컴파일 에러 발생

const의 사용

- 전부 사용하거나 또는 아예 안 쓰도록.
- 만약 함수 수정이 필요없다면
  - const로 매개변수를 보호
  - 그러한 매개변수 전부를 보호
- 이것은 클래스 멤버 함수 매개변수도 포함

```
void welcome(const BankAccount& yourAccount)
{
    cout << "은행 방문을 환영합니다.\n"
         << "고객님 은행 잔고는 아래와 같습니다.\n" ;
    yourAccount.output() ;
}
```

  - void output() const ;로 반드시 선언되어야 함
    - output이 멤버변수를 변경시킨다면 yourAccount의 const가 위배될 수 있으므로

인라인 함수

- 멤버 함수가 아닌 경우:
  - 함수 선언과 함수 정의 앞에 inline 키워드 사용
- 클래스 멤버 함수인 경우:
  - 클래스 정의 부분에서 함수에 대한 구현코드가 위치

    → 자동적으로 inline
- 오직 매우 짧은 함수에 사용
- 실제로 코드가 호출되는 위치에 삽입됨
  - 오버헤드가 줄어듬
  - 매우 효율적이나 오직 짧은 함수에 대해서만 사용하라.

정적 멤버

- 정적 멤버 변수
  - 클래스의 모든 객체가 단 하나의 복사본만 공유
  - 한 객체가 변경하면 → 모든 객체가 변경된 것을 본다.
- "추적"에 유용함
  - 얼마나 자주 멤버 함수가 호출되었는지
  - 얼마나 많은 객체가 존재하는지
- 유형 앞에 static 키워드가 위치

정적 함수

- 멤버 함수도 정적 멤버 가능
  - 만약 객체의 데이터를 참조하지 않고,
    여전히 클래스의 멤버로 두기 원한다면,
    이 함수를 정적 함수로 만듦
  - 클래스 외부에서 호출 가능
  - 클래스의 객체가 아닌 방법으로:
    - 예, Server::getTurn();
  - 뿐만 아니라 클래스의 객체를 통해
    - 기본적인 방법: myObject.getTurn();
- 정적 함수에서는 오직 정적 데이터만 사용 가능!

디스플레이 7.6 정적 멤버

디스플레이 7.6 정적 멤버

```
1   #include <iostream>
2   using namespace std;

3   class Server
4   {
5   public:
6       Server(char letterName);
7       static int getTurn( );
8       void serveOne( );
9       static bool stillOpen( );
10  private:
11      static int turn;
12      static int lastServed;
13      static bool nowOpen;
14      char name;
15  };

16  int Server::turn = 0;
17  int Server::lastServed = 0;
18  bool Server::nowOpen = true;

19  int main( )
20  {
21      Server s1('A'), s2('B');
22      int number, count;
23      do
24      {
25          cout << "How many in your group? ";
26          cin >> number;
27          cout << "Your turns are: ";
28          for (count = 0; count < number; count++)
29              cout << Server::getTurn( ) << ' ';
30          cout << endl;
31          s1.serveOne( );
32          s2.serveOne( );
33      } while (Server::stillOpen( ));

34      cout << "Now closing service.\n";
35      return 0;
36  }
```

```
How many in your group? 3
Your turns are: 1 2 3
Server A now serving 1
Server B now serving 2
How many in your group? 2
Your turns are: 4 5
Server A now serving 3
Server B now serving 4
How many in your group? 0
Your turns are:
Server A now serving 5
Now closing service.
```

디스플레이 7.6 정적 멤버 (2 of 2)

```
39   Server::Server(char letterName) : name(letterName)
40   {/*의도적으로 비워 두었다*/}

41   int Server::getTurn( )
42   {
43       turn++;
44       return turn;
45   }
46   bool Server::stillOpen( )
47   {
48       return nowOpen;
49   }
```

← getTurn이 정적이므로 정적 멤버만이 여기서 참조될 수 있다.

```
50   void Server::serveOne( )
51   {
52       if (nowOpen && lastServed < turn)
53       {
54           lastServed++;
55           cout << "Server " << name
56                << " now serving " << lastServed << endl;
57       }

58       if (lastServed >= turn) //Everyone served
59           nowOpen = false;
60   }
```

벡터(Vector)

- "길이가 커지거나 줄어들 수 있는 배열“
  - C++의 “배열 " 은 크기 변경 불가한 것에 비해
    벡터는 프로그램 실행 중에 크기 변경
  - 표준 템플릿 라이브러리로부터 형성(STL)
  - 배열과 유사하게, 기본형을 가지고 기본형에 해당하는 값들을 저장

- 선언 방법: vector<Base_Type>
  - 템플릿 클래스를 나타냄
    - Base_Type에 어떤 형을 집어넣으면,
      그것을 기본형으로 갖는 벡터를 위한 클래스가 만들어짐

```
vector<int> v;
vector<DayOfYear> d1, d2 ;
```

벡터 사용하기

```
vector<int> v;
```

  - "v는 int형의 벡터"
  - 클래스의 디폴트 생성자 호출되어 빈 벡터 객체가 생성됨
- 원소를 접근하기 위해 배열처럼 인덱스를 가짐
- 원소를 추가할 때는push_back 멤버 함수를 호출해야 함

- 멤버 함수 size(): 현재 원소의 개수를 리턴
- 멤버 함수 capacity()
  - 현재 할당된 메모리 즉 용량을 리턴
    - 용량을 크기 이상이어야. 용량이 부족하면 자동으로 증가됨
- 만약 효율성이 문제된다면:
  - 수동적으로 용량 설정 가능
    - v.reserve(32); // 용량을 32로 설정
    - v.reserve(v.size()+10); // 현재 원소 수보다 10개 많게 용량 설정

디스플레이 7.7 벡터 사용

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;

4   int main( )
5   {
6       vector<int> v;
7       cout << "Enter a list of positive numbers.\n"
8            << "Place a negative number at the end.\n";

9       int next;
10      cin >> next;
11      while (next > 0)
12      {
13          v.push_back(next);
14          cout << next << " added. ";
15          cout << "v.size( ) = " << v.size( ) << endl;
16          cin >> next;
17      }

18      cout << "You entered:\n";
19      for (unsigned int i = 0; i < v.size( ); i++)
20          cout << v[i] << " ";
21      cout << endl;

22      return 0;
23  }
```

Sample Dialogue

```
Enter a list of positive numbers.
Place a negative number at the end.
2 4 6 8 -1
2 added. v.size = 1
4 added. v.size = 2
6 added. v.size = 3
8 added. v.size = 4
You entered:
2 4 6 8
```

요약

- 생성자: 클래스 데이터의 자동 초기화
  - 객체가 선언될 때 호출되는 클래스와 같은 이름을 갖는 특수 멤버함수
- 디폴트 생성자는 매개변수가 없다.
  - 가능하면 항상 정의하도록!
- 클래스 멤버 변수
  - 다른 클래스의 객체가 될 수 있어 초기화 섹션이 요구됨
- Constant call-by-reference 매개 변수
  - 참조에 의한 호출보다 더 효율적
- 매우 짧은 함수는 inline으로 구성하여 효율성 향상
- 정적 멤버 변수는 클래스의 모든 객체에 의해 공유
- 벡터 클래스는 "길이가 커지거나 줄어들 수 있는 배열"

실습

- 분수를 위한 class Fraction을 정의하고 구현하여
  아래 프로그램을 동작하게 하시오.
  - main 그대로 사용할 것
  - 분수의 분모와 분자는 항상 정수
    - 음수인 경우 분모나 분자를 음수로 저장
  - 반드시 파일 분리
    - Fraction.cpp Fraction.h
  - Fraction의 모든 instance는 항상
    약분된 상태로 저장되게 하시오.
    - 힌트: 생성자 Fraction(int d, int n)를 잘 구현

```
#include <iostream>
#include "Fraction.h"
                                    16/15
int main()
{
    Fraction f1, f2(2, 5), f3 ; // f2 = 2/5
    f1.setDenom(3) ;  // Demon = 분모
    f1.setNumer(2) ;  // Numer = 분자
    f3 = f1.add(f2) ; // f1 = 2/3
    f3.print() ;
    return 0 ;
}
```

실습

- 앞서 구현한 행렬 합과 곱을 다음과 같이 class Matrix를 사용하여 구현하시오.
  - main 그대로 사용
  - 파일 분리

```
|  1   2   3 |
|  4   5   6 |
|  7   8   9 |

|  1  -1   0 |
|  0  -1   1 |
| -1   1   0 |

두 행렬의 합은
|  2   1   3 |
|  4   4   7 |
|  6   9   9 |

행렬의 곱은
| -2   0   2 |
| -2  -3   5 |
| -2  -6   8 |
```

```
#include <iostream>
#include "Matrix.h"

int main()
{
    Matrix m1, m2 ;  // 자동으로 3x3 랜덤 행렬 생성.
                     // 각 요소는 -10~10 범위의 값이 되도록
    m1.print() ; m2.print() ;

    Matrix m3 = m1.add(m2) ;
    cout << "두 행렬의 합은 " << endl ;
    m3.print() ;

    Matrix m3 = m1.multi(m2) ;
    cout << "두 행렬의 곱은 " << endl ;
    m3.print() ;
    return 0 ;
}
```