CHAPTER 03

표현식과 문장

학습 목표

- 표현식을 만드는 방법과 표현식의 종류를 알아봅니다.
- 암묵적 변환과 명시적 변환 등 표현식의 자료형 변환을 알아봅니다.
- 우선 순위와 결합 방향을 기반으로 표현식의 평가 순서를 알아봅니다.
- 정수나 부동 소수점과 같은 숫자 자료형의 오버플로우와 언더플로우 현상을 이해합니다.
- 매개변수가 없거나 하나만 있는 간단한 조정자를 사용해서 입력 값을 조작, 출력하는 방법을 알아봅니다.
- 문장을 이해하고 선언문, null 문, 복합문, return 문 등의 기본적인 문장을 학습합니다.
- 프로그램 설계의 3단계(문제 이해하기, 알고리즘 개발하기, 코드 작성하기)를 활용하여 프로그램을 작성해봅니다.

07 [Lab] 응용 프로젝트

이번 절에서는 4개의 프로그램을 설계하고 구현합니다. 다음과 같이 3단계로 이루어지는 '소프트웨어 개발 프로세스'를 사용합니다.

❶ **문제 이해** 문제를 제대로 이해했는지 확인합니다.

❷ **알고리즘 개발** 프로그램을 위한 알고리즘을 개발합니다. 알고리즘을 개발할 때는 여러 가지 도구를 사용할 수 있지만 이번 장의 문제들은 굉장히 간단하므로 글로만 정리합니다.

❸ **코드 작성** 개발한 알고리즘을 기반으로 C++ 코드를 작성합니다.

1 부동 소수점 분해하기

부동 소수점이 주어졌을 때 정수와 소수점 아래 부분을 추출하는 프로그램을 만듭니다.

문제 이해

부동 소수점이 주어지면 이를 정수 부분과 소수점 아래 부분으로 분리합니다. 예를 들어서 123.78이 주어지면 123과 0.78을 출력합니다.

알고리즘 개발

이번 문제의 알고리즘은 굉장히 간단합니다. 숫자를 입력받고 정수 부분과 소수점 아래 부분을 분리한 뒤 원래 숫자, 정수 부분, 소수점 아래 부분을 따로 출력합니다. 단계별로 알고리즘을 정리하면 다음과 같습니다.

```
1. 입력
  a. 숫자 입력받음
2. 처리
  a. 정수 부분 추출
  b. 소수점 아래 부분 추출
3. 출력
  a. 원래 숫자 출력
```

```
b. 정수 부분 출력
c. 소수점 아래 부분 출력
```

코드 작성

프로그램의 모든 요소(입력 부분, 처리 부분, 출력 부분)를 정리했으므로 [프로그램 3-28]에서 코드를 작성합니다.

프로그램 3-28 숫자에서 정수 부분과 소수점 아래 부분을 분리해서 추출하는 프로그램 Prg3-28.cpp

```
1  /*****************************************************************************
2   * 주어진 부동 소수점의 정수 부분과 소수점 아래 부분을                        *
3   * 추출해서 출력하는 프로그램                                                 *
4   *****************************************************************************/
5  #include <iostream>
6  #include <iomanip>
7  using namespace std;
8
9  int main()
10 {
11     // 변수 선언
12     double number;
13     int intPart;
14     double fractPart;
15     // 입력받기
16     cout << "부동 소수점 입력: ";
17     cin >> number;
18     // 처리
19     intPart = static_cast<int>(number);
20     fractPart = number - intPart;
21     // 출력
22     cout << fixed << showpoint << setprecision(2);
23     cout << "원래 값: " << number << endl;
24     cout << "정수 부분: " << intPart << endl;
25     cout << "소수점 아래 부분: " << fractPart;
26     return 0;
27 }
```

실행 결과

```
부동 소수점 입력: 145.72
원래 값: 145.72
정수 부분: 145
소수점 아래 부분: 0.72
```

실행 결과

```
부동 소수점 입력: -546
원래 값: -546.00
정수 부분: -546
소수점 아래 부분: 0.00
```

실행 결과

```
부동 소수점 입력: -0.14
원래 값: -0.14
정수 부분: 0
소수점 아래 부분: -0.14
```

Fixed, showpoint, setprecision 조정자를 사용해서 출력을 만들었습니다. 각각의 실행 결과에서 부동 소수점을 하나씩만 입력했다는 것에 유의하세요. 프로그램이 제대로 만들어졌는지 확인하고 싶다면 출력으로 나온 정수 부분과 소수점 아래 부분을 더했을 때, 원래 숫자가 나오는지 검토하세요. 첫 번째 실행 결과를 예시로 테스트해보면 145 + 0.72 = 145.72이므로 프로그램에 문제가 없다는 것을 알 수 있습니다.

2 정수의 첫 번째 자릿수 추출하기

이번에는 정수가 주어졌을 때 첫 번째 자릿수(가장 오른쪽의 숫자)를 추출하여, 출력하는 프로그램을 설계합니다.

문제 이해

정수가 주어졌을 때, 첫 번째 자릿수를 추출하는 프로그램입니다. 예를 들어서 6759를 입력하면 9가 출력됩니다.

알고리즘 개발

이번 프로그램의 알고리즘은 굉장히 간단합니다. 일단 정수를 입력받아서 첫 번째 자릿수를 추출합니다. 이어서 원래 숫자와 첫 번째 자릿수의 숫자를 출력합니다. 알고리즘은 다음과 같습니다.

```
1. 입력
  a. 정수 입력받음
2. 처리
  a. 첫 번째 자릿수(가장 오른쪽의 숫자) 추출
3. 출력
  a. 원래 숫자 출력
  b. 첫 번째 자릿수 출력
```

코드 작성

이전과 같이 입력, 처리, 출력 부분을 기반으로 프로그램을 만들면 다음과 같습니다. [프로그램 3-29]를 살펴봅니다.

프로그램 3-29 정수의 첫 번째 자릿수 추출하기 Prg3-29.cpp

```
1  /*************************************************************************
2   * 입력된 정수의 첫 번째 자릿수를 추출해서 출력하는 프로그램          *
3   *************************************************************************/
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      // 변수 선언
10     unsigned int givenInt, firstDigit;
11     // 입력받기
12     cout << "양의 정수 입력: ";
13     cin >> givenInt;
14     // 처리
15     firstDigit = givenInt % 10;
16     // 출력
17     cout << "입력한 정수: " << givenInt << endl;
18     cout << "첫 번째 자릿수 추출: " << firstDigit << endl;
19     return 0;
20 }
```

실행 결과

```
양의 정수 입력: 253
입력한 정수: 253
첫 번째 자릿수 추출: 3
```

실행 결과

```
양의 정수 입력: 45672
입력한 정수: 45672
첫 번째 자릿수 추출: 2
```

3 초 단위의 시간을 시, 분, 초 단위로 나누어서 변환하기

초 단위로 주어진 시간이 몇 시간, 몇 분, 몇 초인지 구하는 문제를 설계, 구현합니다.

문제 이해

어떤 일을 하는 데 걸린 시간을 234,572처럼 초 단위의 큰 숫자로 입력받는다고 가정합니다. 이때 이 시간이 몇 시간, 몇 분, 몇 초인지 계산하는 프로그램입니다.

알고리즘 개발

이번 문제도 굉장히 간단한 알고리즘입니다. 초 단위를 나타내는 숫자를 입력받고 이를 시, 분, 초 단위로 변환하여 구한 숫자들을 출력하면 됩니다. 알고리즘은 다음과 같습니다.

```
1. 입력
  a. 초 단위로 시간을 정수 단위로 입력받음
2. 처리
  a. 초 단위를 시 단위로 변환해서 구함
  b. a 과정 후에 남는 초 단위의 시간을 분 단위로 변환해서 구함
  c. b 과정 후에 남은 초 단위를 구함
3. 출력
  a. 입력받은 초 단위를 출력
  b. 2.a에서 구한 시를 출력
  c. 2.b에서 구한 분을 출력
  d. 2.c에서 구한 초를 출력
```

코드 작성

그럼 지금까지 살펴보았던 알고리즘을 기반으로 [프로그램 3-30]에서 코드를 작성합니다.

프로그램 3-30 초 단위의 시간을 시, 분, 초 단위로 나누어서 변환하기 Prg3-30.cpp

```
/*****************************************************************************
 * 초 단위로 입력받은 시간을 시, 분, 초 단위로 나누어서                      *
 * 출력하는 프로그램                                                         *
 *****************************************************************************/
#include <iostream>
using namespace std;

int main()
{
    // 변수 선언
```

```
11      unsigned long duration, hours, minutes, seconds;
12      // 입력받기
13      cout << "초 단위 시간을 양의 정수로 입력: ";
14      cin >> duration;
15      // 처리
16      hours = duration / 3600L;
17      minutes = (duration - (hours * 3600L)) / 60L;
18      seconds = duration - (hours * 3600L) - (minutes * 60);
19      // 출력
20      cout << "입력된 초 단위 시간: " << duration << endl;
21      cout << "결과: ";
22      cout << hours << "시 ";
23      cout << minutes << "분 ";
24      cout << seconds << "초";
25      return 0;
26  }
```

실행 결과

```
초 단위 시간을 양의 정수로 입력: 4000
입력된 초 단위 시간: 4000
결과: 1시 6분 40초
```

실행 결과

```
초 단위 시간을 양의 정수로 입력: 39250
입력된 초 단위 시간: 39250
결과: 10시 54분 10초
```

결과가 출력되었다는 것만으로 답이 제대로 나왔다는 것을 알 수 없습니다. 설계한대로 프로그램이 잘 만들었는지 결과를 테스트하는 과정이 필요합니다. 출력을 반대로 계산해서 입력이 나오는지 확인하는 방법을 사용할 수 있습니다.

```
실행1: (1 * 3600) + (6 * 60) + 40 = 3600 + 360 + 40 = 4000초
실행2: (10 * 3600) + (54 * 60) + 10 = 36000 + 3240 + 10 = 39250초
```

계산했을 때 입력한 값이 나오므로 프로그램에 문제가 없다는 것을 알 수 있습니다.

4 평균과 분산 구하기

이번에는 3개의 정수를 읽고 전체 평균을 구한 뒤, 각각의 분산을 구하는 프로그램을 만듭니다.

문제 이해

평균은 모든 입력 값을 더한 뒤에 개수로 나누어서 구할 수 있습니다. 분산은 특정 값이 평균에서 얼마나 떨어져 있는지를 나타냅니다. 따라서 양수와 음수가 모두 결과로 나올 수 있습니다.

예를 들어서 10, 14, 15라는 숫자가 있다고 가정합니다. 모든 숫자를 더하면 39가 나옵니다. 따라서 평균은 39 / 3 = 13입니다. 분산은 10의 경우 10 − 13 = −3, 14의 경우 14 − 13 = 1, 15의 경우 15 − 13 = 2입니다.

알고리즘 개발

이번 문제는 알고리즘이 약간 복잡합니다. 3개의 숫자를 입력받고 총합과 평균을 구합니다. 이어서 평균을 기반으로 각 숫자의 분산을 구합니다. 알고리즘을 정리하면 다음과 같습니다.

```
1. 입력
  a. 3개의 숫자를 입력받음
2. 처리
  a. 3개의 숫자를 더해서 총합 구함
  b. 총합을 3으로 나누어서 평균 구함
  c. 평균을 기반으로 각 숫자의 분산 구함
3. 출력
  a. 총합 출력
  b. 평균 출력
  c. 각 숫자의 분산 출력
```

코드 작성

그럼 이제 [프로그램 3-31]을 참조하여 코드를 작성해보세요.

프로그램 3-31 총합, 평균, 분산 구하기 Prg3-31.cpp

```
1  /*************************************************************************
2   * 3개의 정수를 입력받고, 이를 더하고 평균을 찾은 뒤                      *
3   * 각 숫자의 분산을 구하는 프로그램                                       *
4   *************************************************************************/
5  #include <iostream>
6  #include <iomanip>
7  using namespace std;
8
9  int main()
10 {
11     // 변수 선언
12     int num1, num2, num3;
13     int sum;
14     double average;
15     double dev1, dev2, dev3;
```

```
16      // 입력받기
17      cout << "첫 번째 숫자 입력: ";
18      cin >> num1;
19      cout << "두 번째 숫자 입력: ";
20      cin >> num2;
21      cout << "세 번째 숫자 입력: ";
22      cin >> num3;
23      // 처리
24      sum = num1 + num2 + num3;
25      average = static_cast<double>(sum) / 3;
26      dev1 = num1 - average;
27      dev2 = num2 - average;
28      dev3 = num3 - average;
29      // 출력
30      cout <<  fixed << setprecision(2);
31      cout << "세 숫자의 합: " << sum << endl;
32      cout << "평균: " << setw(9) << average << endl;
33      cout << "첫 번째 숫자의 분산: " << setw(9) << dev1 << endl;
34      cout << "두 번째 숫자의 분산: " << setw(9) << dev2 << endl;
35      cout << "세 번째 숫자의 분산: " << setw(9) << dev3 << endl;
36      return 0;
37  }
```

실행 결과

```
첫 번째 숫자 입력: 100
두 번째 숫자 입력: 101
세 번째 숫자 입력: 103
세 숫자의 합: 304
평균: 101.33
첫 번째 숫자의 분산: -1.33
두 번째 숫자의 분산: -0.33
세 번째 숫자의 분산: +1.67
```

이전과 마찬가지로 제대로 된 답이 나왔는지 확인합니다. 각 숫자의 분산을 평균에 더하면 원래 값이 나와야 합니다. 참고로 프로그래밍 언어의 내부적인 문제로 완벽하게 딱 떨어지는 답이 아닌 근사값이 나올 수도 있습니다.

- 101.33 + (−1.33) = 100(첫 번째 값)
- 101.33 + (−0.33) = 101(두 번째 값)
- 101.33 + (+1.67) = 103(세 번째 값)

CHAPTER 04

조건문

학습 목표

- 조건 분기를 할 때 관계 표현식과 일치 표현식이 필요한 이유를 이해합니다.
- if 조건문, if-else 조건문, 중첩 if-else 조건문의 사용 방법을 알아봅니다.
- 논리 연산자와 관계·조건 표현식을 사용해서 복잡한 결정을 만드는 방법을 알아봅니다.
- 조건부 표현식을 사용해서 if-else 조건문을 모방하는 방법을 알아봅니다.

05 [Lab] 응용 프로젝트

이번 절에서는 3개의 프로그램을 설계하고 구현합니다. 3장에서 살펴보았던 소프트웨어 개발 단계를 기반으로 문제를 이해하고 알고리즘을 개발한 뒤, 코드를 작성합니다.

1 학생 성적 구하기

A 교수님은 시험 성적 3개에서 최대값과 최소값을 확인하고, 이 두 값의 평균을 학생 성적으로 결정합니다. A 교수님이 성적을 계산할 때 도움을 주는 프로그램을 만듭니다.

문제 이해

우선 입력, 처리, 출력이라는 3가지 과정이 필요합니다. 입력으로 3개의 점수를 입력받고 점수들에서 최대 점수와 최소 점수를 찾습니다. 마지막으로 이 둘의 평균을 구한 뒤 이를 학생의 점수로 설정합니다. 이후의 장을 살펴보면 이를 구현할 수 있는 효율적인 방법이 있지만, 일단 지금까지 배운 조건문만으로 최대 점수와 최소 점수를 찾아봅니다.

알고리즘 개발

다음과 같은 알고리즘을 만들 수 있습니다.

```
1. 입력
  a. score1 입력받기
  b. score2 입력받기
  c. score3 입력받기
2. 처리
  a. 최대 점수(maxScore) 찾기
     ❶ score1이 다른 두 값보다 크면 maxScore는 score1
     ❷ score2가 다른 두 값보다 크면 maxScore는 score2
     ❸ 그렇지 않으면 maxScore는 score3
  b. 최소 점수(minScore) 찾기
     ❶ score1이 다른 두 값보다 작으면 minScore는 score1
     ❷ score2가 다른 두 값보다 작으면 minScore는 score2
     ❸ 그렇지 않으면 minScore는 score3
```

```
  c. 학생의 성적 찾기
     ❶ maxScore와 minScore를 더하기
     ❷ 더한 값을 2로 나누어서 학생의 성적을 구함
3. 출력
  a. 세 성적(score)을 출력
  b. minScore와 maxScore를 출력
  c. 학생의 성적을 출력
```

코드 작성

이제 알고리즘을 기반으로 코드를 작성합니다. 47~52행은 추가 설명이 필요한 부분입니다. 학생의 성적 최대값과 최소값은 정수이며, 결과도 정수로 얻기 원한다고 가정했습니다. 두 정수를 더한 뒤 단순하게 2로 나누면 학생 성적이 내림(예를 들어 85.5가 85가 되는 현상)되는데, 1을 더해서 반올림하게 만든 것입니다.

프로그램 4-15 점수 입력을 기반으로 학생의 성적 찾기 Prg4-15.cpp

```
1  /*************************************************************************
2   * 학생 점수를 3개 입력받고                                              *
3   * 최소값과 최대값을 기반으로 평균을 구해서                              *
4   * 학생의 성적을 찾는 프로그램                                           *
5  *************************************************************************/
6  #include <iostream>
7  using namespace std;
8
9  int main()
10 {
11     // 변수 선언
12     int score1, score2, score3, maxScore, minScore, score;
13     // 입력받기
14     cout << "첫 번째 점수 입력: ";
15     cin >> score1;
16     cout << "두 번째 점수 입력: ";
17     cin >> score2;
18     cout << "세 번째 점수 입력: ";
19     cin >> score3;
20     // 가장 큰 점수 찾기
21     if(score1 > score2 && score1 > score3)
22     {
23         maxScore = score1;
24     }
```

```
25      else if(score2 > score1 && score2 > score3)
26      {
27          maxScore = score2;
28      }
29      else
30      {
31          maxScore = score3;
32      }
33      // 가장 작은 점수 찾기
34      if(score1 < score2 && score1 < score3)
35      {
36          minScore = score1;
37      }
38      else if(score2 < score1 && score2 <= score3)
39      {
40          minScore = score2;
41      }
42      else
43      {
44          minScore = score3;
45      }
46      // 학생 성적을 구하고 올림하기(홀수일 때만 올림)
47      int temp = maxScore + minScore;
48      if(temp % 2 == 1)
49      {
50          temp += 1;
51      }
52      score = temp / 2;
53      // 결과 출력
54      cout << "입력한 점수 = " << score1 << " " << score2 << " " <<
        score3 << endl;
55      cout << "최소 점수와 최대 점수 = ";
56      cout << minScore << " " << maxScore << endl;
57      cout << "학생 성적 = " << score;
58      return 0;
59  }
```

실행 결과

```
첫 번째 점수 입력: 78
두 번째 점수 입력: 92
세 번째 점수 입력: 79
입력한 점수 = 78 92 79
최소 점수와 최대 점수 = 78 92
학생 성적 = 85
```

실행 결과

```
첫 번째 점수 입력: 65
두 번째 점수 입력: 93
세 번째 점수 입력: 60
입력한 점수 = 65 93 60
최소 점수와 최대 점수 = 60 93
학생 성적 = 77
```

2 소득 범위에 따른 세금 구하기

특정 사람의 소득을 기반으로 세금을 계산한 뒤 출력하는 프로그램을 설계합니다.

문제 이해

대부분의 국가에서는 [그림 4-14]처럼 수입의 범위에 따라 다른 세율의 소득세가 적용됩니다.

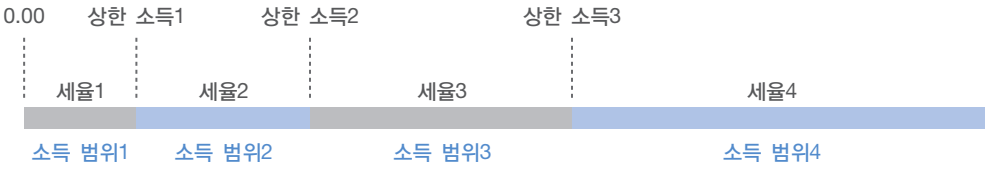

그림 4-14 소득 범위에 따른 소득세

표 4-4 소득세율

소득 범위	세금1	세금2	세금3	세금4
소득 범위1	차이×비율1			
소득 범위2	상한1×비율1	차이×비율2		
소득 범위3	상한1×비율1	상한2×비율2	차이×비율3	
소득 범위4	상한1×비율1	상한2×비율2	상한3×비율3	차이×비율4

총 세금은 '각 소득 범위 안에 있는 세금의 합계'라고 할 수 있습니다. 소득 범위1에 있는 사람은 세금1만큼을 냅니다. 소득 범위2에 있는 사람은 세금1, 세금2의 합계를 냅니다. 소득 범위3에 있는 사람은 세금1, 세금2, 세금3의 합계를 냅니다. 소득 범위4에 있는 사람은 세금1, 세금2, 세금3, 세금4의 합계를 냅니다. 표에서 '차이'라는 것은 현재 소득과 이전 상한 소득과의 차이를 뜻합니다.

알고리즘 개발

이번 알고리즘은 간단합니다. 수입을 입력받은 뒤에 조건문으로 세금을 계산한 뒤 출력합니다. 전체 과정을 그림으로 정리하면 [그림 4-15]와 같습니다.

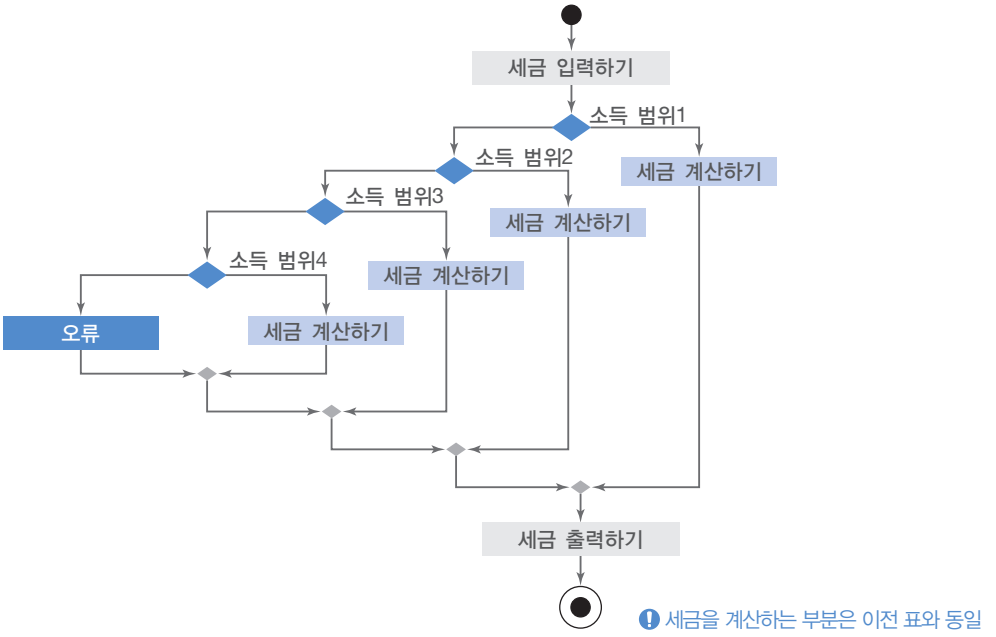

그림 4-15 소득 범위에 따라 세금을 구하는 알고리즘

코드 작성

이제 코드를 작성합니다. 프로그램에 사용되는 소득 범위와 세금 비율이 이후에 바뀔 수도 있습니다. 확장성을 위해서 이러한 요소를 변수로 만들어서 사용합니다. 추가로 소득 범위를 나타내는 4개의 불 자료형의 변수 bracket1~bracket4도 만듭니다. bracket1이 true라면 첫 번째 소득 범위에 있다는 의미입니다. 입력의 유효성을 검사할 수 있게 수입이 음수일 때 42~43행처럼 오류 메시지를 출력하고 곧바로 리턴하게 만들 것입니다. 43행의 리턴이 있어야 46, 47행을 출력을 하지 않고 바로 종료됩니다.

프로그램 4-16 소득 범위에 따른 세금 구하기 Prg4-16.cpp

```
1  /*****************************************************************************
2   * 4가지 서로 다른 소득 범위를 기반으로 세금을 계산해서                      *
3   * 출력하는 프로그램                                                         *
4   *****************************************************************************/
5  #include <iostream>
6  using namespace std;
7
```

```
8  int main()
9  {
10     // 변수 선언
11     double income, tax ;
12     bool bracket1, bracket2, bracket3, bracket4;
13     double limit1 = 10000.00, limit2 = 50000.00, limit3 = 100000.00;
14     double rate1 = 0.05, rate2 = 0.10, rate3 = 0.15, rate4 = 0.20;
15     // 입력받기
16     cout << "수입을 달러로 입력하세요: " ;
17     cin >> income;
18     // 소득 범위 찾기
19     bracket1 = (income <= limit1) && (income >= 0);
20     bracket2 = (income > limit1) && (income <= limit2);
21     bracket3 = (income > limit2) && (income <= limit3);
22     bracket4 = (income > limit3);
23     // 세금 계산
24     if(bracket1)
25     {
26         tax = income * rate1;
27     }
28     else if(bracket2)
29     {
30         tax = limit1 * rate1 + (income - limit1) * rate2 ;
31     }
32     else if(bracket3)
33     {
34         tax = limit1 * rate1 + (limit2 - limit1) * rate2 +
                 (income - limit2) * rate3 ;
35     }
36     else if(bracket4)
37     {
38         tax = limit1 * rate1 + (limit2 - limit1) * rate2 +
                 (limit3 - limit2) * rate3 + (income - limit3) * rate4 ;
39     }
40     else
41     {
42         cout << "입력에 문제가 있습니다.";
43         return 0;
44     }
45     // 수입과 세금 출력
46     cout << "수입 = " << income << endl;
47     cout << "세금 = " << tax;
```

```
48      return 0;
49  }
```

실행 결과

```
수입을 달러로 입력하세요: 8500
수입 = 8500
세금 = 425
```

실행 결과

```
수입을 달러로 입력하세요: 14500
수입 = 14500
세금 = 950
```

실행 결과

```
수입을 달러로 입력하세요: -5
입력에 문제가 있습니다.
```

실행 결과

```
수입을 달러로 입력하세요: 123000
수입 = 123000
세금 = 16600
```

세 번째 실행(−5를 입력한 경우)은 입력이 유효하지 않습니다. 이외의 경우들은 각각의 소득 범위를 테스트한 것입니다. 첫 번째 실행은 소득 범위1에 있는 소득이며, 소득의 5%인 425달러를 납부합니다. 두 번째 실행은 소득 범위2에 있으며 소득 범위1에 속하는 10,000달러의 세금으로 500달러, 소득 범위2에 속하는 남은 4,500달러의 세금으로 450달러를 납부합니다. 각각의 결과가 맞는지 이와 같은 방법으로 확인해보세요.

3 날짜 번호 구하기

한 해의 모든 날에 날짜 번호를 매긴다고 가정합니다. 예를 들어서 1월 1일을 1번째 날이고, 12월 31일은 365번째 날입니다. 몇 월 며칠이 1년에서 몇 번째 날인지 구합니다.

문제 이해

문제를 풀려면 1월부터 12월까지 각각의 달에 날이 몇 개 있는지 미리 지정해야 합니다. 이를 기반으로 몇 월 며칠이 1년에서 몇 번째 날인지 단순하게 더하면 됩니다.

알고리즘 개발

이번 문제의 알고리즘은 크게 2가지로 구분할 수 있습니다. 첫 번째는 몇 월인지 지정했을 때, 해당 월까지의 날짜 수를 구하는 것입니다. 그리고 두 번째는 첫 번째에서 구한 날짜 수와 이번 달의 며칠인지 하는 날짜를 더해서 최종적인 답을 얻는 것입니다. 첫 번째 과정은 switch 조건문으로 구하고 두 번째 과정은 단순한 덧셈으로 구합니다.

break 구문을 사용하지 않는 형태로 switch 조건문을 활용하면 첫 번째 과정을 쉽게 처리할 수 있습니다. switch 조건문의 case 분기를 12월부터 1월까지 거꾸로 잡은 뒤, 각 월의 날짜

를 추가하는 형태로 코드를 작성하면 switch 조건문이 끝났을 때 해당 월까지의 날짜를 쉽게 구할 수 있습니다.

코드 작성

[프로그램 4-17]은 이를 구현한 것입니다. 윤년이라서 2월이 29일이 되는 경우와 입력에 문제가 있는 경우는 무시합니다. 이와 관련된 내용은 이후에 설명합니다.

프로그램 4-17 날 수 구하기 Prg4-17.cpp

```
/*****************************************************************************
 * 몇 월 며칠을 입력받았을 때                                                *
 * 해당 날짜가 올해 몇 번째 날짜인지 구하는 프로그램                         *
 *****************************************************************************/
#include <iostream>
using namespace std;

int main()
{
    // 변수 선언
    int month;
    int day;
    int totalDays = 0;
    // 입력받기
    cout << "몇 월인지 입력하세요: ";
    cin >> month;
    cout << "며칠인지 입력하세요: ";
    cin >> day;
    // 각 월의 날 수
    int m01 = 31;
    int m02 = 28;
    int m03 = 31;
    int m04 = 30;
    int m05 = 31;
    int m06 = 30;
    int m07 = 31;
    int m08 = 31;
    int m09 = 30;
    int m10 = 31;
    int m11 = 30;
    // switch 조건문을 사용해서 특정 달까지의 날 수를 계산
    switch(month)
```

```
33      {
34          case 12 : totalDays += m11;
35          case 11 : totalDays += m10;
36          case 10 : totalDays += m09;
37          case 9  : totalDays += m08;
38          case 8  : totalDays += m07;
39          case 7  : totalDays += m06;
40          case 6  : totalDays += m05;
41          case 5  : totalDays += m04;
42          case 4  : totalDays += m03;
43          case 3  : totalDays += m02;
44          case 2  : totalDays += m01;
45          case 1  : totalDays += 0;
46      }
47      // 최종 결과를 구함
48      totalDays += day;
49      // 결과 출력
50      cout << "올해의 " << totalDays << "번째 날입니다.";
51      return 0;
52  }
```

실행 결과

```
몇 월인지 입력하세요: 1
며칠인지 입력하세요: 23
올해의 23번째 날입니다.
```

실행 결과

```
몇 월인지 입력하세요: 4
며칠인지 입력하세요: 12
올해의 102번째 날입니다.
```

실행 결과

```
몇 월인지 입력하세요: 11
며칠인지 입력하세요: 24
올해의 328번째 날입니다.
```

실행 결과

```
몇 월인지 입력하세요: 12
며칠인지 입력하세요: 31
올해의 365번째 날입니다.
```

CHAPTER 05

반복문

학습 목표

- 반복문을 구성할 때 사용하는 표현식을 이해합니다.
- while 반복문의 기본 문법과 구성할 때 필요한 카운터 제어, 이벤트 제어를 알아봅니다.
- for 반복문의 헤더가 어떤 형태로 작동하는지 파악합니다.
- do-while 반복문을 사용하여 자료의 유효성 검사를 하는 방법을 이해합니다.
- 중첩 반복문의 사용 방법을 익히고 직접 프로그래밍해봅니다.
- 반복과 관련된 구문을 살펴보고 되도록 사용을 피하는 방법을 알아봅니다.

07 [Lab] 응용 프로젝트

1 입력받은 숫자의 합과 곱 구하기

이번 절에서는 조건 분기와 반복을 사용하여 컴퓨터 과학에서 고전적으로 꼽는 문제들을 해결해봅시다. 물론 이후에 배우는 내용을 사용하면 코드를 훨씬 더 효율적으로 작성할 수 있습니다. 하지만 일단은 현재까지 배운 내용들만 활용합니다.

이전에 입력받은 숫자들의 합을 구하는 프로그램을 만들었습니다. 이번 절에서는 다시 한 번 입력받은 숫자들의 합과 곱을 구하는 고전 방법을 알아볼 것입니다. 숫자들의 합과 곱을 구한다는 것은 다음과 같은 의미입니다.

```
합 = 17.0 + 14.4 + ... + 71.2
곱 = 17.0 × 14.4 × ... × 71.2
```

문제 이해

일단 숫자 리스트를 사용자로부터 입력받아야 합니다. 변수 sum(합)과 product(곱)를 초기화한 뒤, 반복문을 사용해서 다음과 같은 형태로 숫자들의 합과 곱을 구할 것입니다.

반복1 반복2 ... 반복n	sum += 17.0 sum += 14.4 ... sum += 71.2	product *= 17.0 product *= 14.4 ... product *= 71.2
■ 초기화	■ sum = 0.0	■ product = 1.0

이때 변수 sum과 product의 초기값이 다르다는 것에 주의하세요. 변수 sum은 0.0으로, 변수 product는 1.0으로 초기화해야 합니다.

알고리즘 개발

프로그램을 실행할 때마다 리스트의 크기가 달라질 수 있으므로 일단 사용자에게 리스트의 크기를 입력받도록 합니다. 이렇게 리스트의 크기를 입력받으면 카운터 제어 반복문을 사용할 수 있습니다. 이벤트 제어 반복문으로도 구현 가능합니다.

- **입력받아야 하는 것** 반복문에 진입하기 전에 무엇을 입력받아야 하는지 생각해봅시다. 카운터 제어 반복문으로 구현할 것이므로 리스트의 크기를 입력받아야 합니다.
- **초기화해야 하는 것** 합과 곱을 나타내는 변수를 만들고 초기화해야 합니다. 또한 카운터 제어 반복문을 사용할 것이므로 카운터도 초기화가 필요합니다.
- **각 반복에서 해야 하는 것** 각 반복에서는 사용자로부터 숫자를 읽어 들이고, 이를 합과 곱을 계산할 때 활용합니다. 또한 카운터 제어 반복문이므로 카운터도 증가시켜서 사용해야 합니다.
- **출력해야 하는 것** 최종적으로 출력해야 하는 것은 모든 숫자의 합과 곱입니다. 반복문이 종료된 이후에 출력합니다.

알고리즘을 정리하면 다음과 같습니다.

```
1. 입력
   a. 변수 size(크기) 입력받기
2. 초기화
   a. 변수 sum(합)을 0.0으로 초기화
   b. 변수 product(곱)를 1.0으로 초기화
   c. 변수 counter를 0으로 초기화
3. counter가 size보다 작은 경우 반복
   a. 숫자 읽어 들이기
   b. 기존의 변수 sum과 읽어 들인 숫자를 합하고, sum에 할당
   c. 기존의 변수 product와 읽어 들인 숫자를 곱하고, product에 할당
   d. 변수 counter 변경
4. 출력
   a. 변수 sum 출력
   b. 변수 product 출력
```

코드 작성

[프로그램 5-18]은 알고리즘을 기반으로 구현한 프로그램입니다. long double 자료형의 숫자를 입력받고 합과 곱을 구합니다. long double 자료형은 범위가 굉장히 넓으므로 오버플로우 또는 언더플로우가 거의 발생하지 않을 것입니다.

프로그램 5-18 입력받은 숫자의 합과 곱 구하기 Prg5-18.cpp

```
/*************************************************************************
 * 리스트의 크기를 처음부터 정했을 때                                    *
 * 리스트 내부 숫자의 합과 곱을 구하는 프로그램                          *
 *************************************************************************/
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    // 변수 선언
    int size;
    long double number;
    long double sum, product;
    // 입력받고 유효성 검사
    do
    {
        cout << "입력할 숫자의 수를 음수가 아닌 정수로 입력하세요: " ;
        cin >> size;
    } while(size < 0);
    // 초기화
    sum = 0;
    product = 1;
    // 처리
    for(int i = 1; i <= size; i++)
    {
        cout << "다음 숫자를 입력하세요: ";
        cin >> number;
        sum += number;
        product *= number;
    }
    // 출력
    cout << fixed << setprecision(2);
    cout << "합 = " << sum << endl;
    cout << "곱 = " << product;
    return 0;
}
```

실행 결과

```
입력할 숫자의 수를 음수가 아닌 정수로 입력하세요: 6
다음 숫자를 입력하세요: 12
다음 숫자를 입력하세요: 13.45
다음 숫자를 입력하세요: 15
다음 숫자를 입력하세요: 22.10
다음 숫자를 입력하세요: 11.34
다음 숫자를 입력하세요: 14
합 = 87.89
곱 = 8494310.92
```

실행 결과

```
입력할 숫자의 수를 음수가 아닌 정수로 입력하세요: 0
합 = 0.00
곱 = 1.00
```

첫 번째 실행은 6개의 숫자를 더하고 곱했습니다. 두 번째 실행은 리스트에 아무 것도 없는 경우(size = 0)입니다. 단순히 초기값을 출력하는데, 리스트에 아무 것도 없을 때는 리스트에 아무 것도 없다고 출력해주는 것이 좋습니다.

2 팩토리얼 구현하기

팩토리얼factorial은 정수 n이 주어졌을 때, 1부터 n까지의 모든 정수를 곱한 값을 의미합니다. 이번에는 팩토리얼을 반복문으로 구현합니다.

문제 이해

팩토리얼을 간단히 표현하면 $1 \times 2 \times 3 \times \cdots \times n$입니다. 이를 n 팩토리얼이라고 부르며 수학에서는 n!이라고 표현합니다.

알고리즘 개발

팩토리얼을 구하는 알고리즘은 이전 문제의 알고리즘과 거의 비슷합니다. 하지만 카운터를 0이 아닌 1부터 시작합니다. 만약 카운터가 0부터 시작합니다면 0이 곱해져서 결과가 무조건 0으로 나옵니다. 또한 n까지 곱해야 하므로 n이 아닌 n + 1이 될 때 반복문을 벗어나야 합니다.

```
1. 입력
   a. 변수 n 입력받기
2. 초기화
```

```
   a. 변수 factorial을 1로 초기화
   b. 변수 counter를 1로 초기화
3. counter가 n + 1보다 작은 경우 반복
   a. 기존의 변수 factorial에 변수 counter를 곱하고 factorial에 할당
   b. 변수 counter 변경
4. 출력
   a. 변수 factorial 출력
```

코드 작성

알고리즘에 따라서 코드를 작성하면 [프로그램 5-19]와 같습니다. 오버플로우가 발생할 수 있으므로, 부호 없는 long 자료형으로 변수 factorial을 선언해서 오버플로우의 위험을 줄였습니다.

프로그램 5-19 팩토리얼 구현하기 Prg5-19.cpp

```
1  /*****************************************************************************
2   * 리스트 내부의 숫자 곱을 활용해서 팩토리얼(n!)을 구하는                    *
3   * 프로그램                                                                  *
4   *****************************************************************************/
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     // 변수 선언
11     int n;
12     unsigned long long factorial;
13     // 입력받기
14     do
15     {
16         cout << "팩토리얼의 크기를 입력하세요: ";
17         cin >> n;
18     } while(n < 0);
19     // 초기화
20     factorial = 1;
21     // 처리
22     for(int i = 1; i < n + 1; i++)
23     {
24         factorial *= i;
25     }
26     // 출력
27     cout << n << "! = " << factorial;
```

```
28      return 0;
29  }
```

실행 결과

```
팩토리얼의 크기를 입력하세요: 0
0! = 1
```

실행 결과

```
팩토리얼의 크기를 입력하세요: 4
4! = 24
```

실행 결과

```
팩토리얼의 크기를 입력하세요: 12
12! = 479001600
```

실행 결과

```
팩토리얼의 크기를 입력하세요: 22
22! = 17196083355034583040
```

실행 결과

```
팩토리얼의 크기를 입력하세요: 30
30! = 9682165104862298112
```

실행 결과를 분석합니다. 첫 번째 실행 결과는 0을 입력했습니다. 0! = 1인데, 이는 기본 정의입니다. 두 번째 실행 결과는 4를 입력했습니다. $4! = 4 \times 3 \times 2 \times 1 = 24$이므로 정상적으로 실행되었습니다. 세 번째 실행 결과는 12!를 의미합니다. 갑자기 숫자가 확 커졌습니다. 이렇게 팩토리얼은 기하급수적으로 늘어납니다.

네 번째와 다섯 번째 실행 결과를 보면 조금 이상하다는 것을 알 수 있습니다. 22!이 30!보다 큽니다. 따라서 이는 오버플로우가 발생다는 것을 의미합니다. 하지만 현재 실행 결과로는 어디에서 오버플로우가 발생했는지 알 수 없습니다. n = 22 이전에서 오버플로우가 발생했을 가능성도 있습니다.

3 거듭제곱 구하기

반복문을 공부할 때 많이 살펴보는 내용으로 숫자의 거듭제곱(b^n)을 구하는 예제가 있습니다. 이때 b를 밑base이라고 부르고 n을 지수exponent라고 부릅니다.

문제 이해

간단하게 밑을 반복해서 곱합니다. 물론 C++은 기본 라이브러리로 거듭제곱과 관련된 함수를 제공합니다. 하지만 이를 직접 구현합니다. 부호 없는 정수를 입력받고, 부호 없는 정수로 결과를 낸다고 가정합니다.

```
거듭제곱 = b^n = b × b ×  b × ... × b × b(n번 반복)
```

알고리즘 개발

알고리즘을 정리하면 다음과 같습니다.

```
1. 입력
   a. 변수 base(밑)를 입력받음
   b. 변수 exponent(지수)를 입력받음
2. 초기화
   a. 변수 power를 1로 초기화
   b. 변수 counter를 0으로 초기화
3. counter가 exponent보다 작은 경우 반복
   a. 기존의 변수 power와 변수 base를 곱하고, power에 할당
   b. 카운터 변경
4. 출력
   a. 변수 power 출력
```

코드 작성

알고리즘을 구현하면 [프로그램 5-20]과 같으며 오버플로우 상황을 처리하는 코드도 추가했습니다.

프로그램 5-20 거듭제곱 구하기 Prg5-20.cpp

```
1  /*****************************************************************************
2   * 반복문을 사용해서 밑을 지수의 수만큼 곱해서                                *
3   * 거듭제곱을 구하는 프로그램                                                 *
4   *****************************************************************************/
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     // 변수 선언
11     int base, exponent;
12     unsigned long int power, temp;
13     bool overflow;
14     // 유효성 검사하면서 입력받기
15     do
16     {
17         cout << "밑을 음수가 아닌 정수로 입력하세요: " ;
18         cin >> base;
19     } while(base < 0);
```

```
20      // 유효성 검사하면서 입력받기
21      do
22      {
23          cout << "지수를 음수가 아닌 정수로 입력하세요: " ;
24          cin >> exponent;
25      } while(exponent < 0);
26      // 초기화
27      power = 1;
28      temp = power;
29      overflow = false;
30      // 처리
31      for(int i = 1; (i <= exponent) && (!overflow); i++)
32      {
33          power *= base;
34          if(power / base  != temp)
35          {
36              overflow = true; // 반복문 종료
37          }
38          temp = power;
39      }
40      // 출력
41      if(overflow)
42      {
43          cout << "오버플로우가 발생했습니다. 작은 값으로 시도하세요." ;
44      }
45      else
46      {
47          cout << base << "^" << exponent << " = " << power;
48      }
49      return 0;
50  }
```

실행 결과

```
밑을 음수가 아닌 정수로 입력하세요: 22
지수를 음수가 아닌 정수로 입력하세요: 5
22^5 = 5153632
```

실행 결과

```
밑을 음수가 아닌 정수로 입력하세요: 5
지수를 음수가 아닌 정수로 입력하세요: 10
5^10 = 9765625
```

실행 결과

```
밑을 음수가 아닌 정수로 입력하세요: 5
지수를 음수가 아닌 정수로 입력하세요: 28
오버플로우가 발생했습니다. 작은 값으로 시도하세요.
```

일단 사용자가 음수를 입력하지 않도록 do-while 반복문을 사용했습니다. 이렇게 유효성 검사를 했고, 마지막 실행 결과인 5^28은 오버플로우가 일어났다는 것에 주목하세요.

4 최소값과 최대값 찾기

컴퓨터 과학에서는 리스트 내부의 숫자 중에서 최소값과 최대값을 찾는 활동도 많이 합니다. 이번에는 이를 구현합니다.

문제 이해

예를 들어서 17, 14, 12, 71이라는 숫자 리스트가 있다고 가정합니다. 반복문에 들어가기 전에 일단 변수 smallest를 +infinity로 초기화합니다. 그리고 반복 내부에서 현재 반복하고 있는 숫자가 변수 smallest 보다 작은 경우, smallest에 할당합니다.

```
//항목
17
14
12
71
```

```
//smallest = infinity
smallest의_값_infinity > 현재_단계의_숫자_17 → smallest = 17로 할당
smallest의_값_17 > 현재_단계의_숫자_14 → smallest = 14로 할당
smallest의_값_14 > 현재_단계의_숫자_12 → smallest = 12로 할당
smallest의_값_12 > 현재_단계의_숫자_71 → 아무 것도 하지 않음
결과 12
```

알고리즘 개발

알고리즘을 정리하면 다음과 같습니다.

```
1. 입력
   a. 변수 size(크기) 입력받기
2. 초기화
   a. 변수 smallest(최소값)를 표현할 수 있는 가장 큰 값(+infinity)으로 초기화
   b. 변수 largest(최대값)를 표현할 수 있는 가장 작은 값(-infinity)으로 초기화
   c. 변수 counter를 0으로 초기화
3. counter가 size보다 작은 경우 반복
   a. 숫자 읽어 들이기
```

```
   b. 현재 숫자가 변수 smallest보다 작으면, 변수 smallest에 현재 숫자 할당
   c. 현재 숫자가 변수 largest보다 크면, 변수 largest에 현재 숫자 할당
   d. 카운터 변경
4. 출력
   a. 변수 smallest 출력
   b. 변수 largest 출력
```

코드 작성

[프로그램 5-21]은 지금까지 설명한 알고리즘을 조합해서 정수 리스트 내부에서 최소값과 최대값을 찾는 프로그램입니다.

프로그램 5-21 리스트 내부에서 최소값과 최대값 찾기 Prg5-21.cpp

```
1  /*************************************************************************
2   * 정수 리스트의 크기를 미리 알고 있을 때                                *
3   * 정수 리스트 내부의 최소값과 최대값을 구하는 프로그램                  *
4   *************************************************************************/
5  #include <iostream>
6  #include <limits>  // 자료형의 범위를 확인하기 위한 헤더 파일
7  using namespace std;
8
9  int main()
10 {
11     // 변수 선언
12     int size;
13     int number, smallest, largest;
14     // 초기화
15     smallest = numeric_limits<int>::max();
16     largest = numeric_limits<int>::min();
17     // 입력받기
18     do
19     {
20         cout << "음수가 아닌 수로 리스트의 크기를 입력하세요: ";
21         cin >> size;
22     } while(size <= 0);
23     // 처리
24     for(int i = 1; i <= size; i++)
25     {
26         cout << "다음 숫자를 입력하세요: ";
27         cin >> number;
28         if(number < smallest)
```

```
29          {
30              smallest = number;
31          }
32          if(number > largest)
33          {
34              largest = number;
35          }
36      }
37      // 결과 출력
38      cout << "최소값 = " << smallest << endl;
39      cout << "최대값 = " << largest << endl;
40      return 0;
41  }
```

실행 결과

```
음수가 아닌 수로 리스트의 크기를 입력하세요: 6
다음 숫자를 입력하세요: 12
다음 숫자를 입력하세요: -3
다음 숫자를 입력하세요: 14
다음 숫자를 입력하세요: 15
다음 숫자를 입력하세요: 27
다음 숫자를 입력하세요: -7
최소값 = -7
최대값 = 27
```

실행 결과

```
음수가 아닌 수로 리스트의 크기를 입력하세요: 3
다음 숫자를 입력하세요: 1
다음 숫자를 입력하세요: 87
다음 숫자를 입력하세요: 45
최소값 = 1
최대값 = 87
```

5 적어도(any) 또는 모두(all) 조건에 충족하는 숫자 찾기

컴퓨터 과학에서는 리스트가 어떤 기준에 충족하는지 확인하는 상황이 많습니다. 이때 리스트 내부에 있는 항목 중에 적어도 하나가 기준에 충족하는지 찾는 것을 any(적어도) 문제라고 부릅니다. 리스트 내부에 있는 항목 모두가 기준에 충족하는지 찾는 것을 all(모두) 문제라고 부릅니다.

문제 이해

일반적으로 반복문을 사용해서 항목들이 기준에 맞는지 하나하나 확인합니다. 적어도 하나가 그런지 확인하는 경우(any 문제)에는 기준에 맞는 항목을 찾는 순간 반복을 벗어나면 됩니다. 반면 모두 그런지 확인하는 경우(all 문제)에는 기준에 맞지 않는 항목이 하나라도 발견되는 순간 반복을 벗어나면 됩니다. [그림 5-16]을 참고하세요.

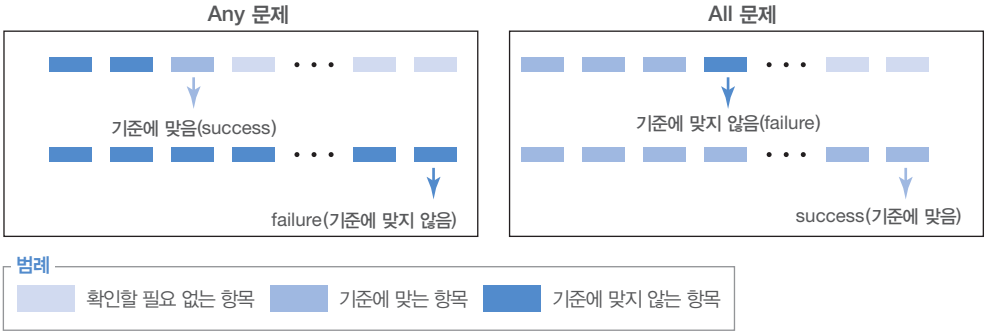

그림 5-16 any 문제와 all 문제

알고리즘 개발

기준에 맞는 항목을 찾거나 또는 기준에 맞지 않는 항목을 찾았을 때 검색을 중지해야 하므로, 일단 카운터 제어(센티넬 또는 EOF와 조합)와 플래그 제어를 모두 활용합니다. 플래그로 사용할 변수의 이름을 success라고 합니다. any 문제는 success를 false로 초기화하고, all 문제는 success를 true로 초기화합니다. 그리고 any 문제는 반복문을 돌다가 기준에 맞는 항목을 찾으면 success를 true로 변경하고, all 문제는 기준에 맞지 않는 항목을 찾으면 success를 false로 변경합니다. 따라서 기준에 맞지 않는 항목을 만나지 않으면 success가 true로 유지됩니다.

이러한 조건을 조금 더 간단하게 만들 수 있게 반복문을 종료할 수 있는 조건을 생각해봅시다. 드 모르간의 법칙을 적용해서 정리하면 다음과 같습니다.

■ 문제	■ 종료 조건	■ 유지 조건
any 문제	(요소가 더 없는 경우) or (success)	(요소가 더 있는 경우) and (!success)
all 문제	(요소가 더 없는 경우) or (!success)	(요소가 더 있는 경우) and (success)

전체적인 알고리즘을 정리하면 다음과 같습니다.

1. 입력
   a. 변수 size(크기) 입력받기
2. 초기화
   a. 변수 success = false
   b. 변수 counter = 0
3. (counter < size or !success) 조건으로 반복
   a. 다음 항목을 읽어 들임
   b. 기준에 맞는지를 success에 할당
   c. counter++
4. if success
   a. 적어도 하나가 기준에 맞는다고 출력
5. else
   a. 하나도 기준에 맞는 것이 없다고 출력

■ any 문제

1. 입력
   a. 변수 size(크기) 입력받기
2. 초기화
   a. 변수 success = true
   b. 변수 counter = 0
3. (counter < size or success) 조건으로 반복
   a. 다음 항목을 읽어 들임
   b. 기준에 맞는지를 success에 할당
   c. counter++
4. if success
   a. 모두 기준에 맞는다고 출력
5. else
   a. 모두 기준에 맞지는 않는다고 출력

■ all 문제

코드 작성

두 프로그램이 굉장히 비슷하므로 [프로그램 5-22]처럼 any 문제 하나만 구현합니다. 이 코드는 유효성 검사 반복문, 검색 반복문, 기준을 만족하는 항목이 있는지 확인하는 조건문으로 구성됩니다.

프로그램 5-22 리스트에 있는 항목이 기준을 만족하는지 확인하기 Prg5-22.cpp

```
1  /*****************************************************************************
2   * 리스트 내부에 있는 요소 중 적어도 하나가 7로                                *
3   * 나눌 수 있는지 확인하는 프로그램                                            *
4   *****************************************************************************/
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     // 선언
11     bool success = false;
12     int size;
13     int item;
14     // 입력받기
15     do
16     {
17         cout << "리스트의 크기를 입력하세요: ";
```

```
18          cin >> size;
19      } while(size < 0);
20      // 처리
21      for(int i = 0; (i < size) && (!success); i++)
22      {
23          cout << "다음 숫자를 입력하세요: ";
24          cin >> item;
25          success = (item % 7 == 0);
26      }
27      // 조건 분기
28      if(success)
29      {
30          cout << item << "은/는 7로 나눌 수 있습니다." << endl;
31      }
32      else
33      {
34          cout << "리스트 내부에 7로 나눌 수 있는 숫자가 없습니다." << endl;
35      }
36      return 0;
37  }
```

실행 결과

```
리스트의 크기를 입력하세요: 5
다음 숫자를 입력하세요: 12
다음 숫자를 입력하세요: 32
다음 숫자를 입력하세요: 28
28은/는 7로 나눌 수 있습니다.
```

실행 결과

```
리스트의 크기를 입력하세요: 5
다음 숫자를 입력하세요: 6
다음 숫자를 입력하세요: 12
다음 숫자를 입력하세요: 15
다음 숫자를 입력하세요: 17
다음 숫자를 입력하세요: 22
리스트 내부에 7로 나눌 수 있는 숫자가 없습니다.
```

CHAPTER 06
함수

학습 목표

- 함수의 기본 개념을 이해하고 이를 사용하여 큰 작업을 작은 작업으로 나눌 때 장점을 알아봅니다.
- 함수의 선언, 정의, 호출 과정을 이해합니다.
- 라이브러리 함수와 사용자 정의 함수를 구분합니다.
- 자료 교환(자료 전달, 자료 리턴)할 때 사용하는 메커니즘을 이해합니다.
- 같은 이름을 가지지만 시그니처가 다른 함수를 여러 개 만들 수 있는 오버로딩을 이해합니다.
- 프로그램 내부에 있는 객체가 가질 수 있는 수명을 알아보고, 자동 변수와 정적 변수를 이해합니다.
- 함수를 사용한 프로그램을 설계합니다.

07 [Lab] 응용 프로젝트

1 정기 예금의 미래 가치 찾기

이번 절에서는 2가지 프로그램을 만들어보면서, 함수를 어떻게 사용하는지 알아봅니다. 모두 경제와 관련 있는데, 컴퓨터 과학에서 자주 언급되는 고전적인 예시를 이용합니다. 바로 정기 예금의 미래 가치와 정기 적금의 미래 가치를 찾는 예시입니다.

문제 이해

정기 예금의 미래 가치는 일반적으로 다음 공식을 사용합니다.

$$미래\_가치 = 투자한\_가치 \times (1 + 이율)^{기간}$$

이때 이율rate은 한 주기의 이율이고, 기간term은 그 기간의 반복 횟수를 나타냅니다. 따라서 (1 + 이율)은 1원을 투자한 경우, 다음 주기 때 1원의 가치라고 할 수 있습니다. 이때 경제학 용어로 (1 + 이율)을 요소factor라고 부릅니다. 그리고 (요소^{기간})은 기간만큼 흐른 뒤의 가치입니다. 경제학 용어로 (요소^{기간})을 승수multiplier라고 부릅니다. 이러한 용어들을 변수로 활용할 것이므로 정리하면 다음과 같습니다.

$$요소 = (1 + 이율)$$
$$승수 = 요소^{기간}$$
$$미래\_가치 = 투자한\_가치 \times 승수$$

따라서 미래 가치를 찾으려면, 1원의 미래 가치(승수)를 찾은 뒤에 이를 투자 금액에 곱합니다.

알고리즘 개발

■ **구조 차트**

이번 예시는 [그림 6-28]과 같은 구조로 만듭니다. 3가지 함수(input, process, output)를 갖고, 이 함수들 내부에서 추가로 함수들을 더 호출할 것입니다.

input 함수에서는 getInput 함수를 사용해서 투자한 가치, 이율, 기간을 입력받습니다. 그리고 process 함수에서는 findMultiplier 함수를 사용해서 승수를 구한 뒤에, 이를 투자한 가치에 곱해서 미래 가치를 구합니다. 마지막으로 output 함수에서는 printData 함수와 printResult 함수를 사용해서 입력받은 값들과 최종적인 결과를 출력합니다.

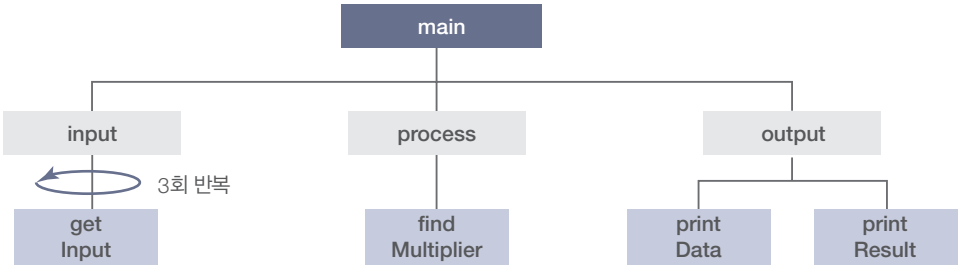

그림 6-28 구조 차트

■ main 함수

main 함수에서는 input, process, output 함수를 호출합니다.

■ input 함수

input 함수는 void 함수로 만들 것입니다. 대신 참조로 전달을 사용해서 변수 3개(투자한 가치, 이율, 기간)에 값을 할당에서 main 함수에 전달합니다. 입력을 받기 위해서 별도의 함수 getInput 함수를 만들고 활용합니다.

■ process 함수

process 함수에서는 미래 가치를 계산합니다. 매개변수를 5개 가질 것인데, input 함수에서 입력받은 3개의 값은 값으로 전달 메커니즘을 사용해 전달합니다. 그리고 마지막 2개는 승수와 미래 가치를 참조로 전달을 사용해서 전달합니다. 이어서 내부적으로 findMultiplier 함수를 호출해서 승수를 계산하고, 모든 변수를 사용해서 미래 가치를 계산합니다. 자세한 공식은 문제 이해 부분을 참고하세요.

■ 출력 함수

printData 함수와 printResult 함수를 출력 함수라고 부릅니다. 출력 함수 내부에서는 사용자의 입력과 최종적인 결과를 출력합니다.

코드 작성

[프로그램 6-26]은 main 함수와 입력 함수, 처리 함수, 출력 함수를 모두 포함한 코드입

니다. getInput 함수가 입력 함수, findMultiplier 함수가 처리 함수, printData 함수와 printResult 함수가 출력 함수입니다.

프로그램 6-26 정기 예금의 미래 가치 찾기 Prg6-26.cpp

```
/*************************************************************************
 * 함수를 사용해서 정기 예금의                                           *
 * 미래 가치를 찾는 프로그램                                             *
 *************************************************************************/
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

// 주요 함수 선언
void input(double& invest, double& rate, double& term);
void process(double invest, double rate, double term,
             double& multiplier, double& futureValue);
void output(double invest, double rate, double term,
            double multiplier, double futureValue);
// 부가 함수 선언
double getInput(string message);
double findMultiplier(double rate, double period);
void printData(double invest, double rate, double term);
void printResult(double multiplier, double value);

int main()
{
    // 변수 선언
    double invest, rate, term;          // 입력 변수
    double multiplier, futureValue;     // 출력 변수
    // 주요 함수 호출
    input(invest, rate, term);
    process(invest, rate, term, multiplier, futureValue);
    output(invest, rate, term, multiplier, futureValue);
    return 0;
}
/*************************************************************************
 * input 함수는 getInput 함수를 호출해서 3개의 입력을 받음              *
 * 참조로 전달을 사용해서 값을 main 함수로 전달                          *
 * 함수가 종료되면 입력들이                                              *
 * invest, rate, term에 저장                                             *
 *************************************************************************/
```

```
void input(double& invest, double& rate, double& term)
{
    invest = getInput("투자 금액을 입력하세요: ");
    rate = getInput("1년 마다의 이율을 입력하세요: ");
    term = getInput("몇 년을 넣을지 입력하세요: ");
}
/*************************************************************************
 * process 함수는 findMultiplier 함수를 사용해서 승수를 계산          *
 * 이어서 미래 가치를 계산                                             *
 * 참조로 전달로 받은 변수들로 값을 리턴                              *
 *************************************************************************/
void process(double invest, double rate, double term,
             double& multiplier, double& futureValue)
{
    multiplier = findMultiplier(rate, term);
    futureValue = multiplier * invest;
}
/*************************************************************************
 * output 함수는 printData 함수를 호출해서 3개의 값을 출력            *
 * printResult 함수를 호출해서 계산한 결과를 출력                     *
 *************************************************************************/
void output(double invest, double rate, double term,
            double multiplier, double futureValue)
{
    printData(invest, rate, term);
    printResult(multiplier, futureValue);
}
/*************************************************************************
 * getInput 함수는 사용자로부터 입력을 받는 함수                      *
 * 매개변수로 사용자에게 어떤 자료를 입력해달라고 요구할지,           *
 * 메시지를 문자열 자료형의 객체로 전달받음                           *
 * 그리고 입력이 양수인지 확인                                        *
 * 최종적으로 입력받은 값을 input 함수로 리턴                         *
 *************************************************************************/
double getInput(string message)
{
    double input;
    do
    {
        cout << message;
        cin >> input;
    } while(input < 0.0);
```

```
81      return input;
82  }
83  /*****************************************************************************
84   * findMultiplier 함수는 값으로 전달 메커니즘으로 이율, 기간을 전달받음  *
85   * 이어서 요소(factor)를 계산한 뒤,                                    *
86   * 이를 기간만큼 제곱해서 승수를 계산하고 리턴하는 함수                *
87   *****************************************************************************/
88  double findMultiplier(double rate, double term)
89  {
90      double factor = 1 + rate/100;
91      return pow(factor, term);
92  }
93  /*****************************************************************************
94  * printData 함수는 사용자로부터 입력받은                                *
95  * 투자 금액(invest), 이율(rate), 기간(term)을 출력하는 함수             *
96  * 이 함수는 부가 작용(화면에 출력)만 하는 void 함수                     *
97  *****************************************************************************/
98  void printData(double invest, double rate, double term)
99  {
100     cout << endl << "투자 정보" << endl;
101     cout << "투자 금액: " << fixed << setprecision(2) << invest << endl;
102     cout << "이율: " << rate << fixed << setprecision(2);
103     cout << "% 연 마다" << endl;
104     cout << "기간: " << term << "년" << endl << endl;
105 }
106 /*****************************************************************************
107  * printResult 함수는 프로그램의 최종적인 결과라고 할 수 있는          *
108  * 승수(multiplier)와 미래 가치(futureValue)를 출력                     *
109  * 이 함수는 부가 작용(화면에 출력)만 하는 void 함수                    *
110  *****************************************************************************/
111 void printResult(double multiplier, double futureValue)
112 {
113     cout << "투자의 승수 = " << fixed << setprecision(8);
114     cout << multiplier << endl;
115     cout << "미래 가치 = " << fixed << setprecision(2);
116     cout << futureValue << endl;
117 }
```

실행 결과

```
투자 금액을 입력하세요: 360000
1년 마다의 이율을 입력하세요: 5
몇 년을 넣을지 입력하세요: 30
투자 정보
투자 금액: 360000.00
이율: 5.00% 연 마다
기간: 30.00년

투자의 승수 = 4.32194238
미래 가치 = 1555899.26
```

결과를 보면 승수multiplier가 4.32194238이라는 것을 알 수 있습니다. 5% 이율로 30년 동안 1원을 넣으면, 최종적으로 4.32194238원이 된다는 뜻입니다.

2 정기 적금의 미래 가치 찾기

[프로그램 6-26]에서는 목돈을 한 번에 넣고, 만기 때에 원금과 이자를 받는 정기 예금을 살펴보았습니다. 이번에는 매 주기 특정 일에 금액을 계속해서 넣고, 만기 때에 원금과 이자를 받는 정기 적금 예시를 만듭니다.

문제 이해

❶ 특정 주기마다 한 번도 빠지지 않고 적금을 넣는다고 가정합니다.

❷ 투자금을 넣었을 때부터 넣은 만큼만 복리로 불어난다고 가정합니다.

알고리즘 개발

앞의 두 가정을 기반으로 이전 프로그램에서 승수를 변경하기만 하면, 정기 적금의 미래 가치를 구할 수 있습니다. 주기마다 넣은 돈이 계속해서 누적되어 복리로 증가하므로, 첫 번째 주기에 넣은 돈은 n주기 동안, 두 번째 주기에 넣은 돈은 n-1주기 동안, 세 번째 주기에 넣은 돈은 n-2주기 동안 늘어납니다. 그리고 마지막 주기에 넣은 돈은 1주기 동안 늘어납니다. 따라서 정리하면 다음과 같습니다.

$$승수 = (1 + 인수)^{n} + (1 + 인수)^{n-1} + \ldots + (1 + 인수)^{1}$$

중고등학교 때 배웠던 수열을 사용하면 이를 하나의 식으로 계산할 수 있지만, 단순하게 반복문으로 다음과 같이 구할 수도 있습니다.

```
double findMultiplier(double rate, double term)
{
    double multiplier = 0;
    double factor = 1 + rate/100;
    for(int i = term; i >= 0; i--)
    {
        multiplier += pow(factor, i);
    }
    return multiplier;
}
```

코드 작성

지금까지 살펴본 내용을 기반으로 이전에 만들었던 예금 프로그램을 적금 프로그램으로 변경합니다.

프로그램 6-27 정기 적금의 미래 가치 찾기 Prg6-27.cpp

```
1  /*************************************************************************
2   * 함수들을 활용해서                                                     *
3   * 정기 적금의 미래 가치를 찾는 프로그램                                 *
4   *************************************************************************/
5  #include <iostream>
6  #include <iomanip>
7  #include <cmath>
8  #include <string>
9  using namespace std;
10
11 // 주요 함수 선언
12 void input(double& invest, double& rate, double& term);
13 void process(double invest, double rate, double term,
14              double& multiplier, double& futureValue);
15 void output(double invest, double rate, double term,
16             double multiplier, double futureValue);
17 // 부가 함수 선언
18 double getInput(string message);
19 double findMultiplier(double rate, double period);
20 void printData(double invest, double rate, double term);
21 void printResult(double multiplier, double value);
22
23 int main()
24 {
```

```
    // 변수 선언
    double invest, rate, term;            // 입력 변수
    double multiplier, futureValue;       // 출력 변수
    // 주요 함수 호출
    input(invest, rate, term);
    process(invest, rate, term, multiplier, futureValue);
    output(invest, rate, term, multiplier, futureValue);
    return 0;
}
/*************************************************************************
 * input 함수는 getInput 함수를 호출해서 3개의 입력을 받음              *
 * 참조로 전달을 사용해서 값을 main 함수로 전달                          *
 * 함수가 종료되면 입력들이 invest, rate, term에 저장됨                  *
 *************************************************************************/
void input(double& invest, double& rate, double& term)
{
    invest = getInput("정기 적금에 투자할 금액을 입력하세요: ");
    rate = getInput("1년 마다의 이율을 입력하세요: ");
    term = getInput("몇 년을 넣을지 입력하세요: ");
}
/*************************************************************************
 * process 함수는 findMultiplier 함수를 사용해서 승수를 계산            *
 * 이어서 미래 가치를 계산                                               *
 * 참조로 전달로 받은 변수들로 값을 리턴                                 *
 *************************************************************************/
void process(double invest, double rate, double term,
             double& multiplier, double& futureValue)
{
    multiplier = findMultiplier(rate, term);
    futureValue = multiplier * invest;
}
/*************************************************************************
 * output 함수는 printData 함수를 호출해서 3개의 값을 출력하고,         *
 * printResult 함수를 호출해서 계산한 결과를 출력                        *
 *************************************************************************/
void output(double invest, double rate, double term,
            double multiplier, double futureValue)
{
        printData(invest, rate, term);
        printResult(multiplier, futureValue);
}
```

```
66  /*********************************************************************
67   * getInput 함수는 사용자로부터 입력을 받는 함수                    *
68   * 매개변수로 사용자에게 어떤 자료를 입력해달라고 요구할지          *
69   * 메시지를 문자열 자료형의 객체로 전달받음                         *
70   * 그리고 입력이 양수인지 확인                                      *
71   * 최종적으로 입력받은 값을 input 함수로 리턴                       *
72   *********************************************************************/
73  double getInput(string message)
74  {
75      double input;
76      do
77      {
78          cout << message;
79          cin >> input;
80      } while(input < 0.0);
81      return input;
82  }
83  /*********************************************************************
84   * 이번 예시의 findMultiplier 함수는                                *
85   * 복리 적금 공식을 활용해서 승수를 계산                            *
86   * 최종적으로 처리한 값을 process 함수로 리턴                       *
87   *********************************************************************/
88  double findMultiplier(double rate, double term)
89  {
90      double multiplier = 0;
91      double factor = 1 + rate/100;
92      for(int i = term; i > 0 ; i--)
93      {
94          multiplier += pow(factor, i);
95      }
96      return multiplier;
97  }
98  /*********************************************************************
99   * printData 함수는 사용자로부터 입력받은                           *
100  * 투자 금액(invest), 이율(rate), 기간(term)을 출력하는 함수         *
101  *********************************************************************/
102 void printData(double invest, double rate, double term)
103 {
104     cout << endl << "정기 적금 정보" << endl;
105     cout << "정기 적금 투자 금액: " << fixed;
106     cout << setprecision(2) << invest << endl;
107     cout << "이율: " << rate << fixed << setprecision(2);
```

```
108     cout << "% 연 마다" << endl;
109     cout << "기간: " << term << "년" << endl << endl;
110 }
111 /*****************************************************************************
112  * printResult 함수는 프로그램의 최종적인 결과라고 할 수 있는            *
113  * 승수(multiplier)와 미래 가치(futureValue)를 출력                       *
114  *****************************************************************************/
115 void printResult(double multiplier, double futureValue)
116 {
117     cout << "투자 결과" << endl;
118     cout << "투자의 승수 = " << fixed << setprecision(8);
119     cout << multiplier << endl;
120     cout << "미래 가치 = " << fixed << setprecision(2);
121     cout << futureValue << endl;
122 }
```

실행 결과

```
정기 적금에 투자할 금액을 입력하세요: 360000
1년 마다의 이율을 입력하세요: 5
몇 년을 넣을지 입력하세요: 30

정기 적금 정보
정기 적금 투자 금액: 360000.00
이율: 5.00% 연 마다
기간: 30.00년

투자 결과
투자의 승수 = 69.76078988
미래 가치 = 25113884.36
```

CHAPTER 08

배열

학습 목표

- 같은 자료형의 데이터 타입이 나열된 배열을 알아봅시다.
- 배열을 생성하고 요소에 접근하는 방법을 이해합니다.
- 배열의 연산을 알아봅니다.
- 1차원 배열의 타입, 용적, 크기라는 3가지 속성을 이해합니다.
- 1차원 배열의 선언과 초기화를 어떻게 하는지 파악합니다.
- 2차원 혹은 3차원 배열을 뜻하는 다차원 배열의 구조를 파악합니다.

04 [Lab] 응용 프로젝트

1 빈도 배열과 히스토그램 만들기

이번 절에서는 컴퓨터 과학에서 배열을 활용하는 전형적인 문제들을 풀어봅니다. 배열은 어떤 상황이 발생하는 횟수를 셀 때 활용할 수 있습니다. 이렇게 발생 빈도를 세는 배열을 빈도 배열frequency array이라고 부릅니다. 그리고 이러한 빈도 배열을 활용하면 히스토그램histogram도 만들 수 있습니다. 히스토그램은 빈도 배열을 그림으로 나타낸 것입니다. [그림 8-19]는 어떤 배열을 기반으로 빈도 배열을 만드는 예시입니다.

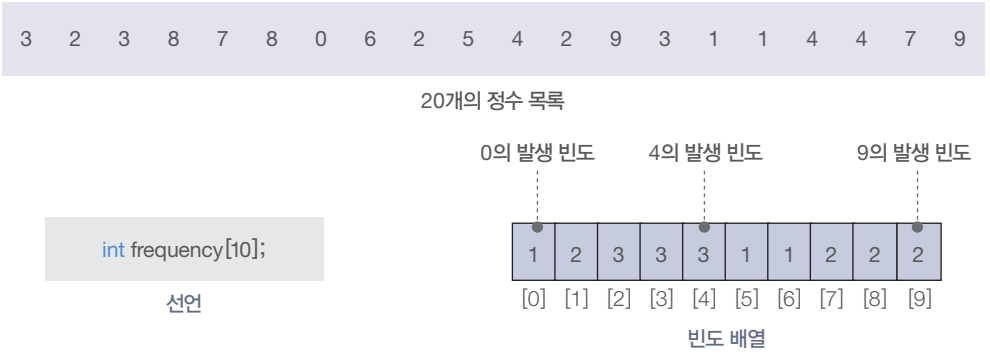

그림 8-19 숫자 목록과 빈도 배열

0~9 범위의 숫자를 수집했다고 가정합니다. 참고로 [그림 8-19]에서는 20개의 숫자를 수집하지만, 수집한 숫자의 수에는 제한이 없습니다. 이러한 숫자 중에 0이 몇 개 있는지, 1이 몇 개 있는지, 2가 몇 개 있는지를 알아봅시다.

숫자가 몇 번씩 등장하는지 빈도를 저장할 수 있게 10개의 요소를 갖는 배열을 만듭니다. 이렇게 빈도를 세기 위한 배열을 빈도 배열이라고 부릅니다. 빈도 배열을 모두 만들면, 숫자가 몇 번씩 등장하는지 알 수 있습니다.

수집한 숫자의 수가 굉장히 많을 것이라고 가정하여 파일로부터 숫자를 읽어 들입니다. 따라서 그림에서 숫자 목록을 파일의 내용이라고 가정하면 됩니다. 파일에서 정수를 하나씩 읽고 빈도 배열을 만들면 되는데, 예를 들어서 숫자 4를 읽으면, 빈도 배열의 네 번째 인덱스에 1을 더합니다. 빈도 배열의 인덱스는 정수이므로, 숫자는 정수만 사용할 수 있습니다.

문제 이해

일단 frequency라는 배열을 만듭니다. frequency 배열의 각 인덱스는 파일에서 읽어 들인 숫자의 빈도를 셀 때 활용할 것입니다. 예를 들어서 파일에서 숫자 6을 읽어 들이면, frequency[6]의 값을 증가시킵니다. 따라서 frequency 배열의 크기는 숫자의 범위와 같아야 합니다. 우리는 숫자의 범위를 0~9로 설정할 것이므로, frequency 배열의 인덱스는 [0]~[9] 범위면 됩니다. 그리고 파일에서 x라는 값을 읽어 들이고, frequency[x]를 1만큼 더해주면 빈도를 셀 수 있습니다. 배열이 모두 만들어지면, 이를 막대 그래프처럼 출력합니다.

알고리즘 개발

다음과 같은 알고리즘을 만들 수 있습니다.

1. 빈도 배열을 선언하고 모든 값을 0으로 초기화합니다.
2. 숫자가 적혀 있는 파일을 엽니다. 정상적으로 열렸는지 확인하고, 그렇지 않다면 프로그램을 즉시 중단합니다.
3. 숫자가 적혀 있는 파일에서 숫자를 하나하나 읽어 들입니다. 우리가 지정한 범위(0~9)의 숫자를 읽으면, 빈도 배열에서 해당 숫자의 빈도를 증가시킵니다. 범위를 벗어난 숫자를 읽을 경우에는 무시합니다.
4. 빈도 배열에 있는 내용을 하나하나 출력합니다. 이때 시각적으로 내용을 확인할 수 있게, 별 기호(*)를 빈도만큼 출력해서 히스토그램을 만듭니다.

코드 작성

프로그램 8-14 Frequency와 Histogram 만들기 Prg8-14.cpp

```
1  /*****************************************************************************
2   * 파일에서 0~9 범위의 정수를 읽어 들이고                                    *
3   * 빈도 배열과 히스토그램을                                                  *
4   * 만들어서 출력하는 프로그램                                                *
5   *****************************************************************************/
6  #include <iostream>
7  #include <fstream>
8  #include <iomanip>
9  using namespace std;
10
11 int main()
12 {
13     // 선언과 초기화
14     const int CAPACITY = 10;
```

```
15      int frequencies[CAPACITY] = {0};
16      ifstream integerFile;
17      // 정수 파일 열기
18      integerFile.open("integerFile.dat");
19      if(!integerFile)
20      {
21          cout << "숫자 파일을 열 수 없습니다." << endl;
22          cout << "프로그램을 중단합니다.";
23          return 0;
24      }
25      // 파일에서 정수 읽어 들이고, 빈도 배열 생성
26      int data;
27      int size = 0;
28      while(integerFile >> data)
29      {
30          if(data >= 0 && data <= 9)
31          {
32              size++;
33              frequencies[data]++;
34          }
35      }
36      // 정수 파일 닫기
37      integerFile.close();
38      // 빈도 배열과 히스토그램 출력
39      cout << "파일 안에 " <<  size << "개의 유효한 데이터가 있습니다." << endl;
40
41      for(int i = 0; i < 10; i++)
42      {
43          cout << setw(3) << i << " ";
44
45          for(int f = 1; f <= frequencies[i] ; f++)
46          {
47              cout << '*' ;
48          }
49          cout << " " << frequencies[i] << endl;
50      }
51      return 0;
52  }
```

실행 결과

```
파일 안에 202개의 유효한 데이터가 있습니다.
0 ********************** 22
1 **** 4
2 ******************** 19
3 ************* 13
4 ******************** 20
5 ********************************** 33
6 **************************** 28
7 **************************** 28
8 ************************* 25
9 ********** 10
```

각 줄에 있는 별 기호의 수는 대응되는 정수의 수입니다. 즉 히스토그램을 만든 것입니다. 다음은 실행에 사용한 파일입니다. 파일 내부에는 205개의 정수가 적혀 있으며, 이 중에서 파란색으로 표시한 3개만 0~9의 범위에 있지 않습니다. 따라서 프로그램이 205개의 정수를 모두 읽어 들이지만, frequency 배열에는 202개의 정수만 들어갑니다.

```
1 3 2 2 5 7 3 2 8 0 6 4 6 7 0 7 8 5 4 2 3 0 6 7 5 8 5 4 8 9 6 5 5 9 2 3 5 2 6 7 8
0 6 4 6 7 0 7 8 5 4 2 13 0 6 7 5 8 5 4 8 9 6 7 0 7 8 5 4 2 3 0 6 7 5 8 5 4 8 9 6 5
2 7 0 7 8 5 4 2 3 0 6 7 5 8 5 4 8 9 6 15 5 9 7 0 7 8 5 4 2 3 0 6 7 5 8 9 6 5 6 6 4
5 9 7 0 7 8 5 4 2 3 0 6 7 5 8 9 6 5 6 6 4 2 7 0 7 8 5 4 2 3 0 6 7 5 8 5 4 8 9 6 5
1 3 2 2 5 7 3 2 8 0 6 4 6 17 0 7 8 5 4 2 3 0 0 6 4 6 7 0 7 8 5 4 2 3 0 6 7 5 8 1 1
```

2 선형 변환하기

이전에 언급했던 것처럼 선형 변환에서 2차원 배열을 1차원 배열로 변환할 때는 행 방향으로 변환할 수도 있고, 열 방향으로 변환할 수도 있습니다. [그림 8-20]은 행 방향 선형 변환과 열 방향 선형 변환을 나타냅니다.

문제 이해

N개의 행과 M개의 열이 있는 2차원 배열을, N×M개의 요소를 갖는 1차원 배열로 변환하면 됩니다.

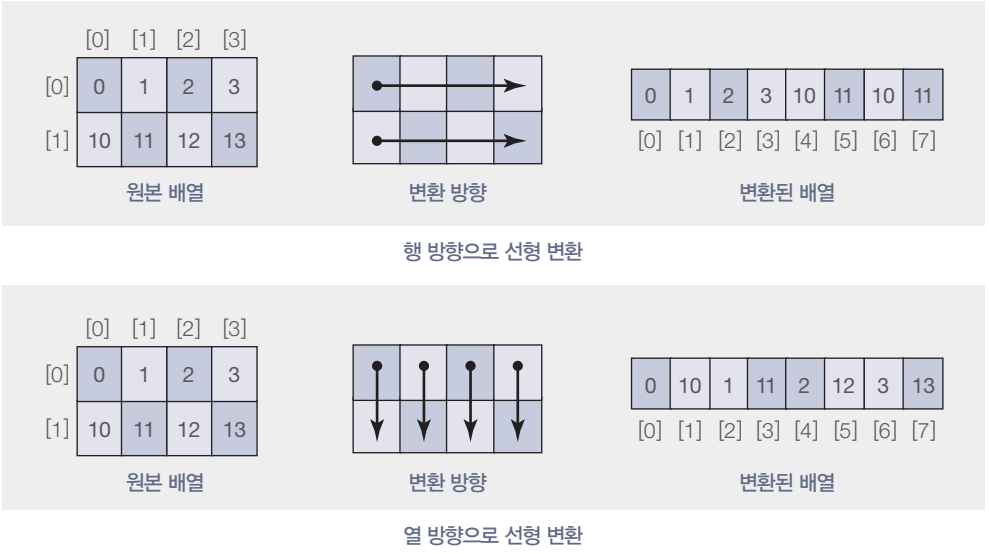

그림 8-20 선형 변환

알고리즘 개발

일단 2차원 배열을 선언하고 초기화합니다. 작은 배열이라면 단순하게 값을 코드에 입력하면 되고, 큰 배열이라면 파일에서 데이터를 읽는 것이 좋습니다. 다음과 같은 함수 4개를 만들어봅시다.

```
1. 행 방향으로 선형 변환하는 함수
2. 열 방향으로 선형 변환하는 함수
3. 2차원 배열을 출력하는 함수
4. 1차원 배열을 출력하는 함수
```

main 함수의 역할은 굉장히 간단합니다. 일단 2차원 배열을 선언하고 초기화합니다. 이어서 1번 함수와 2번 함수를 호출해서 1차원 배열을 만든 뒤, 3번 함수와 4번 함수를 호출해서 모든 배열을 출력합니다.

코드 작성

프로그램 8-15 배열 선형 변환하기 Prg8-15.cpp

```
1  /*****************************************************************************
2   * 2차원 배열을 열 방향과 행 방향으로                                        *
3   * 선형 변환해서                                                             *
4   * 1차원 배열로 만든 뒤 출력하는 프로그램                                    *
5   *****************************************************************************/
```

```
6  #include <iostream>
7  #include <iomanip>
8  using namespace std;
9
10 /*****************************************************************************
11  * rowTransform 함수는 2차원 배열을                                          *
12  * 행 방향으로 선형 변환해서 1차원 배열로 생성                               *
13  * 첫 번째 매개변수는 변경하지 않게 const 한정자를 붙임                      *
14  *****************************************************************************/
15 void rowTransform(const int originArray[][4],
16                   int rowSize, int rowArray[])
17 {
18     int i = 0;
19     int j = 0;
20     for(int k = 0; k < 8; k++)
21     {
22         rowArray[k] = originArray[i][j];
23         j++;
24         if(j > 3)
25         {
26             i++;
27             j = 0;
28         }
29     }
30 }
31 /*****************************************************************************
32  * colTransform 함수는 2차원 배열을                                          *
33  * 열 방향으로 선형 변환해서 1차원 배열로 생성                               *
34  * 첫 번째 매개변수는 변경하지 않게 const 한정자를 붙임                      *
35  *****************************************************************************/
36 void colTransform(const int originArray[][4],
37                   int rowSize, int colArray[])
38 {
39     int i = 0;
40     int j = 0;
41     for(int k = 0; k < 8; k++)
42     {
43         colArray[k] = originArray[i][j];
44         i++;
45         if(i > 1)
46         {
47             j++;
```

```
48              i = 0;
49          }
50      }
51  }
52  /*************************************************************************
53   * 2차원 배열을 매개변수로 받아서 출력                                    *
54   * 매개변수는 변경하지 않게 const 한정자를 붙임                           *
55   *************************************************************************/
56  void printTwoDimensional(const int twoDimensional[][4],
57                           int rowSize)
58  {
59      for(int i = 0; i < rowSize; i++)
60      {
61          for(int j = 0; j < 4; j++)
62          {
63              cout << setw(4) << twoDimensional[i][j];
64          }
65          cout << endl;
66      }
67      cout << endl;
68  }
69
70  /*************************************************************************
71   * 1차원 배열을 매개변수로 받아서 출력                                    *
72   * 매개변수는 변경하지 않게 const 한정자를 붙임                           *
73   *************************************************************************/
74  void printOneDimensional(const int oneDimensional[], int size)
75  {
76      for(int i = 0; i < size; i++)
77      {
78          cout << setw(4) << oneDimensional[i];
79      }
80      cout << endl;
81  }
82
83  int main( )
84  {
85      // 배열 선언하고 초기화
86      int originArray[2][4] = {{0, 1, 2, 3}, {10, 11, 12, 13}};
87      int rowArray[8];
88      int colArray[8];
89      // 2개의 함수를 호출하여 배열 변환
```

```
90      rowTransform(originArray, 2, rowArray);
91      colTransform(originArray, 2, colArray);
92      // 2차원 배열 출력
93      cout << "   원본 배열   " << endl;
94      printTwoDimensional(originArray, 2);
95      // 행 방향으로 선형 변환하고 결과 출력
96      cout << "행 방향으로 선형 변환한 결과: ";
97      printOneDimensional(rowArray, 8);
98      // 열 방향으로 선형 변환하고 결과 출력
99      cout << "열 방향으로 선형 변환한 결과: ";
100     printOneDimensional(colArray, 8);
101     return 0;
102 }
```

실행 결과

```
 원본 배열
  0   1   2   3
 10  11  12  13

행 방향으로 선형 변환한 결과:       0   1   2   3  10  11  12  13
열 방향으로 선형 변환한 결과:       0  10   1  11   2  12   3  13
```

CHAPTER 09

참조, 포인터, 메모리 관리

학습 목표

- 참조 변수와 참조 자료형을 이해합니다.
- 메모리 위치의 주소를 나타내는 포인터를 알아봅니다.
- 포인터 자료형과 포인터 변수를 알아봅니다.
- 배열과 포인터의 관계를 파악합니다.
- 효율적인 메모리 관리를 위해 메모리 구성을 알아봅니다.
- 프로그램이 활용하는 4개의 메모리 영역을 알아봅니다.

05 [Lab] 응용 프로젝트

1 Course 클래스 만들기

이번 절에서는 일부 데이터 멤버가 힙 영역에 만들어지는 클래스 2개를 만듭니다. 우선 대학교에서 각 과목에 대한 통계를 만들 때 활용할 수 있는 Course 클래스를 만듭니다. 과목에 따라서 학생의 수가 다를 수 있으므로 학생들의 점수를 모두 추적하려면 과목별로 학생의 수를 크기로 갖는 배열이 필요합니다. Course 클래스라는 클래스는 관리자가 만들어서 관리하고, 공용 인터페이스만 교수들에게 제공할 것입니다. 관리자의 입장과 교수의 입장 모두가 되어서 Course 클래스를 만들고 활용합니다.

데이터 멤버

[그림 9-30]은 Course 클래스의 데이터 멤버를 나타낸 것입니다.

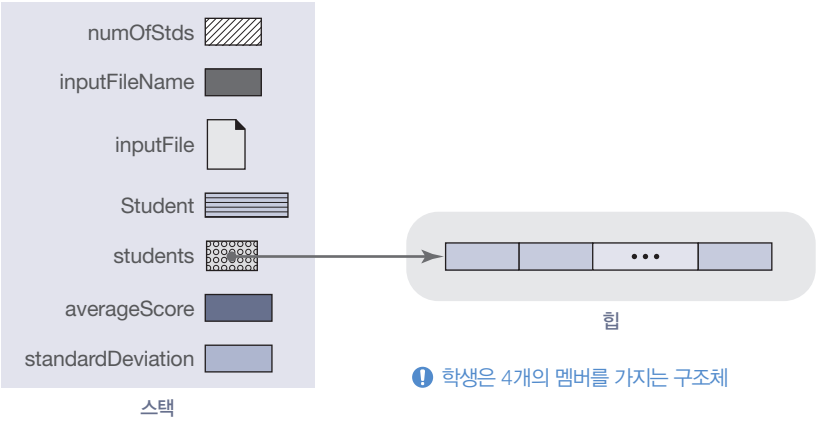

그림 9-30 Course 클래스의 데이터 멤버

학생별로 학번identity, 점수score, 등급grade, 편차deviation를 가지고 있고 이를 구조체를 사용해서 구현합니다. 따라서 힙에 만들어지는 배열은 Student 구조체의 배열이 됩니다.

멤버 함수

public 멤버 함수로 생성자와 소멸자만 구성하고, 이외의 모든 멤버 함수는 private 멤버 함수

로 지정하여 생성자와 소멸자에서 호출하게 만듭니다.

입력 파일

입력 파일은 다음과 같은 데이터를 담고 있습니다. 첫 번째 열에는 학번, 두 번째 열에는 해당 과목의 점수가 적혀 있습니다.

```
1000  88
1001  100
1002  92
1003  77
1004  54
1005  82
1006  67
1007  95
1008  93
1009  100
```

인터페이스, 구현, 애플리케이션 파일

[프로그램 9-16]부터 [프로그램 9-18]까지는 인터페이스, 구현, 애플리케이션 파일입니다.

프로그램 9-16 인터페이스 파일(course.h) Prg9-16.cpp

```
1  /*****************************************************************************
2   * private 멤버 함수로 생성자를 도와주는                                     *
3   * 헬퍼 함수(도우미 함수)를 정의                                             *
4   * 모든 처리는 생성자에서 하고                                               *
5   * 소멸자에서는 힙에 할당한 메모리 영역을 제거하고                           *
6   * 파일을 닫는 프로그램                                                      *
7   *****************************************************************************/
8  #ifndef COURSE_H
9  #define COURSE_H
10 #include <iostream>
11 #include <fstream>
12 using namespace std;
13
14 class Course
15 {
16     private:
17         int numOfStds;
18         const char* inputFileName;
```

```
19          ifstream inputFile;
20          struct Student {int id; int score; char grade;
21                          double deviation;};
22          Student* students;
23          double averageScore;
24          double standardDeviation;
25          void getInput();
26          void setGrades();
27          void setAverage();
28          void setDeviations();
29          void printResult() const;
30
31      public:
32          Course(int numOfStds, const char* inputFileName);
33          ~Course();
34  };
35  #endif
```

프로그램 9-17 구현 파일(course.cpp) Prg9-17.cpp

```
1   /*************************************************************************
2    * 모든 private 멤버 함수와 public 멤버 함수를                          *
3    * 정의하는 구현 파일                                                   *
4    *************************************************************************/
5   #include "course.h"
6   #include <iomanip>
7   #include <cmath>
8
9   /*************************************************************************
10   * 생성자에서는 파일을 열고,                                            *
11   * 힙 배열을 할당한 뒤                                                  *
12   * 여러 헬퍼 함수를 활용                                                *
13   * 배열을 초기화한 뒤                                                   *
14   * 등급, 평균, 표준 편차를                                              *
15   * 구하고 출력하는 프로그램                                             *
16   *************************************************************************/
17  Course::Course(int num, const char* ifn)
18  :numOfStds(num), inputFileName(ifn)
19  {
20      inputFile.open(inputFileName);
21      students = new Student[numOfStds];
22      getInput();
```

```
23      setGrades();
24      setAverage();
25      setDeviations();
26      printResult();
27  }
28  /*************************************************************************
29   * 소멸자에서는 포인터를 사용해서                                        *
30   * 힙에 생성한 배열을 제거하고,                                          *
31   * 생성자에서 열었던 파일을 닫음                                         *
32   *************************************************************************/
33  Course::~Course()
34  {
35      delete[] students;
36      inputFile.close();
37  }
38  /*************************************************************************
39   * getInputs 함수는 입력 파일에서                                        *
40   * 학생들의 학번과 성적을 읽어 들임                                      *
41   *************************************************************************/
42  void Course::getInput()
43  {
44      for(int i = 0; i < numOfStds; i++)
45      {
46          inputFile >> students[i].id;
47          inputFile >> students[i].score;
48      }
49  }
50  /*************************************************************************
51   * setGrades 함수는 배열과 학생들의 성적을 기반으로                      *
52   * 학생들의 등급을 계산                                                  *
53   *************************************************************************/
54  void Course::setGrades()
55  {
56      char charGrades[] =
57              {'F', 'F', 'F',  'F', 'F', 'F', 'D', 'C', 'B', 'A', 'A'};
58      for(int i = 0; i < numOfStds; i++)
59      {
60          int index = students[i].score / 10;
61          students[i].grade = charGrades[index];
62      }
63  }
64  /*************************************************************************
```

```
 * setAverage 함수는 배열 내부에 있는 학생들의 성적을                    *
 * 기반으로 평균을 계산                                                *
 ************************************************************************/
void Course::setAverage()
{
    int sum = 0;
    for(int i = 0; i < numOfStds; i++)
    {
        sum += students[i].score;
    }
    averageScore = static_cast<double>(sum) / numOfStds;
}
/************************************************************************
 * setDeviations 함수는 학생들의 점수와 평균을                          *
 * 기반으로 표준 편차를 계산                                            *
 ************************************************************************/
void Course::setDeviations()
{
    standardDeviation = 0.0;
    for(int i = 0; i < numOfStds; i++)
    {
        students[i].deviation = students[i].score - averageScore;
        standardDeviation += pow(students[i].deviation, 2);
    }
    standardDeviation = sqrt(standardDeviation) / numOfStds;
}
/************************************************************************
 * printResult 함수에서는                                              *
 * 모든 정보를 출력                                                    *
 ************************************************************************/
void Course::printResult() const
{
    cout << endl;
    cout << " ID   점수  등급   편차" << endl;
    cout << "---- ----- ----- -------" << endl;
    for(int i = 0; i < numOfStds; i++)
    {
        cout << setw(4) << noshowpoint << noshowpos;
        cout << right << students[i].id;
        cout << setw(14) << noshowpoint << noshowpos;
        cout << right << students[i].score;
        cout << setw(10) << right << students[i].grade;
```

```
107         cout << fixed << setw(20) << right << setprecision(2);
108         cout << showpoint << showpos;
109         cout << students[i].deviation << endl;
110     }
111     cout << "평균 점수: " << fixed << setw(4);
112     cout << setprecision(2) << averageScore << endl;
113     cout << "표준 편차: " << standardDeviation;
114 }
```

프로그램 9-18 애플리케이션 파일(app.cpp) Prg9-18.cpp

```
1  /*************************************************************************
2   * 애플리케이션 파일에서는                                               *
3   * 단순하게 학생 수와 학생의 학번과 이름이 기록된                        *
4   * 파일의 경로를 매개변수로 전달해서 클래스를 인스턴스화                 *
5   * 모든 처리는 클래스의 생성자에서 진행                                  *
6   *************************************************************************/
7  #include "course.h"
8
9  int main()
10 {
11     // Course 객체 인스턴스화
12     Course course(10, "scores.dat");
13     return 0;
14 }
```

실행 결과

```
ID   점수  등급   편차
---- ----- ----- -------
1000   88     B    +3.20
1001  100     A   +15.20
1002   92     A    +7.20
1003   77     C    -7.80
1004   54     F   -30.80
1005   82     B    -2.80
1006   67     D   -17.80
1007   95     A   +10.20
1008   93     A    +8.20
1009  100     A   +15.20
평균 점수: +84.80
표준 편차: +4.51
```

2 Matrix 클래스 만들기

컴퓨터 프로그래밍에서는 2차원 배열을 사용해서 행렬을 구현할 수 있습니다. 이번에는 2차원 배열을 기반으로 행렬 관련 연산 처리를 하는 Matrix 클래스를 만듭니다.

연산

행렬에 덧셈, 뺄셈, 곱셈이라는 3가지 연산을 정의합니다. 행렬 나눗셈은 행렬을 반전하는 등의 복잡한 처리가 필요하므로 생략합니다.

■ 덧셈

[그림 9-31]처럼 행의 수와 열의 수가 같은 행렬끼리 더할 수 있습니다. 또한 행렬을 더하고 나면 같은 수의 행과 열을 가진 행렬이 나옵니다.

그림 9-31 행렬 2개 더하기

덧셈은 다음과 같이 같은 위치에 있는 값을 더해서 같은 위치에 넣습니다.

```
A = a1 + a2
...
F = f1 + f2
```

■ 뺄셈

뺄셈도 덧셈과 비슷합니다. 다만 이번에는 같은 위치에 있는 값을 빼면 됩니다.

그림 9-32 행렬 2개 빼기

```
A = a1 - a2
...
F = f1 - f2
```

■ **곱셈**

곱셈의 경우는 첫 번째 피연산자 행렬의 열 수와 두 번째 피연산자 행렬의 행 수가 같을 때 할 수 있습니다. 결과로 첫 번째 피연산자의 행 수와 두 번째 피연산자의 열 수를 가진 행렬이 만들어집니다.

역자 행렬의 크기를(행×열)이라고 나타낼 때, (A×B)*(B×C)처럼 연결되는 B가 같을 때 곱셈을 할 수 있습니다. 또한 결과로는 (A×C) 크기의 행렬이 나옵니다.

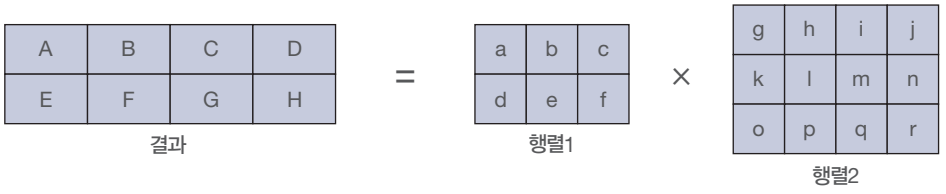

그림 9-33 행렬 곱하기

각각의 요소는 다음과 같은 곱으로 만들어집니다.

```
A = a * g + b * k + c * o
B = a * h + b * l + c * p
...
H = d * j + e * n + f * r
```

인터페이스, 구현, 애플리케이션 파일

그럼 인터페이스 파일, 구현 파일, 애플리케이션 파일을 만듭니다. [프로그램 9-19]는 인터페이스 파일입니다.

프로그램 9-19 인터페이스 파일(matrix.h) Prg9-19.cpp

```
1  /*****************************************************************************
2   * Matrix 클래스의 인터페이스 파일                                           *
3   * 행렬의 크기와 힙에 생성할 행렬을                                          *
4   * 가리킬 포인터만 private 멤버로 선언                                       *
5   * 이외의 모든 멤버 함수는 public 멤버로 선언                                *
6   * 생성자는 힙에 배열을 생성하고, 소멸자는 힙에서 배열을 제거                *
7   * setup 멤버 함수는 행렬을 랜덤한 값으로 초기화                             *
8   * 이외에도 addition, subtraction,                                           *
9   * multiplication, print 멤버 함수를 선언                                    *
10  *****************************************************************************/
11 #include <iostream>
12 #ifndef MATRIX_H
```

```
13 #define MATRIX_H
14 #include <cmath>
15 #include <cstdlib>
16 #include <iomanip>
17 #include <cassert>
18 using namespace std;
19
20 // Matrix 클래스 정의
21 class Matrix
22 {
23     private:
24         int rowSize;
25         int colSize;
26         int** ptr;
27     public:
28         Matrix(int rowSize, int colSize);
29         ~Matrix();
30         void setup();
31         void add(const Matrix& second, Matrix& result) const;
32         void subtract(const Matrix& second, Matrix& result) const;
33         void multiply(const Matrix& second, Matrix& result) const;
34         void print() const;
35 };
36 #endif
```

[프로그램 9-20]은 구현 파일입니다. setup 멤버 함수는 행렬을 랜덤 값으로 채우는 함수이며 행렬을 테스트해보고자 만들었습니다. 실제로 사용할 때는 파일 등으로부터 데이터를 읽어 들일 것입니다.

프로그램 9-20 구현 파일(matrix.cpp) Prg9-20.cpp

```
1  /*************************************************************************
2   * 인터페이스 파일에 선언된                                              *
3   * 모든 함수를 정의하는                                                  *
4   * 인터페이스 파일                                                       *
5   * 이전에 언급한 연산들을 구현                                           *
6   *************************************************************************/
7  #include "matrix.h"
8
9  // 생성자: 힙에 메모리 영역을 잡음
10 Matrix::Matrix(int r, int c)
```

```
: rowSize(r), colSize(c)
{
    ptr = new int*[rowSize];
    for(int i = 0; i < rowSize; i++)
    {
        ptr[i] = new int[colSize];
    }
}
// 소멸자: 힙에 있는 메모리 영역을 해제
Matrix::~Matrix()
{
    for(int i = 0; i < rowSize; i++)
    {
        delete[] ptr[i];
    }
    delete[] ptr;
}
// 행렬의 요소를 랜덤하게 초기화하는 함수
void Matrix::setup()
{
    for(int i = 0; i < rowSize; i++)
    {
        for(int j = 0; j < colSize; j++)
        {
            ptr[i][j] = rand() % 5 + 1;
        }
    }
}
// 두 행렬을 더하는 함수
void Matrix::add(const Matrix& second, Matrix& result) const
{
    assert(second.rowSize == rowSize && second.colSize == colSize);
    assert(result.rowSize == rowSize && result.colSize == colSize);

    for(int i = 0; i < rowSize; i++)
    {
        for(int j = 0; j < second.colSize; j++)
        {
            result.ptr[i][j] = ptr[i][j] + second.ptr[i][j];
        }
    }
}
```

```
53 // 두 행렬을 빼는 함수
54 void Matrix::subtract(const Matrix& second, Matrix& result) const
55 {
56     assert(second.rowSize == rowSize && second.colSize == colSize);
57     assert(result.rowSize == rowSize && result.colSize == colSize);
58     for(int i = 0; i < rowSize; i++)
59     {
60         for(int j = 0; j < second.colSize; j++)
61         {
62             result.ptr[i][j] = ptr[i][j] - second.ptr[i][j];
63         }
64     }
65 }
66 // 두 행렬을 곱하는 함수
67 void Matrix::multiply(const Matrix& second, Matrix& result) const
68 {
69     assert(colSize == second.rowSize);
70     assert(result.rowSize = rowSize);
71     assert(result.colSize = second.colSize);
72     for(int i = 0; i < rowSize; i++)
73     {
74         for(int j = 0; j < second.colSize; j++)
75         {
76             result.ptr[i][j] = 0;
77             for(int k = 0; k < colSize; k++)
78             {
79                 result.ptr[i][j] += ptr[i][k] * second.ptr[k][j];
80             }
81         }
82     }
83 }
84 // 행렬의 요소를 출력하는 함수
85 void Matrix::print() const
86 {
87     for(int i = 0; i < rowSize; i++)
88     {
89         for(int j = 0; j < colSize; j++)
90         {
91             cout << setw(5) << ptr[i][j];
92         }
93         cout << endl;
94     }
```

```
95      cout << endl;
96  }
```

[프로그램 9-21]은 Matrix 클래스를 테스트하는 애플리케이션입니다.

프로그램 9-21 애플리케이션 파일(app.cpp) Prg9-21.cpp

```
1   /*************************************************************************
2    * Matrix 클래스의 객체를 몇 개 만들고                                    *
3    * 연산을 테스트하는 프로그램                                             *
4    *************************************************************************/
5   #include "matrix.h"
6
7   int main()
8   {
9       // matrix1 인스턴스화
10      cout << "matrix1" << endl;
11      Matrix matrix1(3, 4);
12      matrix1.setup();
13      matrix1.print();
14      // matrix2 인스턴스화
15      cout << "matrix2" << endl;
16      Matrix matrix2(3, 4);
17      matrix2.setup();
18      matrix2.print();
19      // matrix3 인스턴스화
20      cout << "matrix3" << endl;
21      Matrix matrix3(4, 2);
22      matrix3.setup();
23      matrix3.print();
24      // matrix1 + matrix2를 구하고 결과 출력
25      cout << "matrix1 + matrix2의 결과" << endl;
26      Matrix addResult(3, 4);
27      matrix1.add(matrix2, addResult);
28      addResult.print();
29      // matrix1 - matrix2를 구하고 결과 출력
30      cout << "matrix1 - matrix2의 결과" << endl;
31      Matrix subResult(3, 4);
32      matrix1.subtract(matrix2, subResult);
33      subResult.print();
34      // matrix1 * matrix3을 구하고 결과 출력
```

```
35      cout << "matrix1 * matrix3의 결과" << endl;
36      Matrix mulResult(3, 2);
37      matrix1.multiply(matrix3, mulResult);
38      mulResult.print();
39      return 0;
40  }
```

실행 결과

```
matrix1
    4    2    3    1
    4    1    2    3
    5    2    3    3

matrix2
    1    5    4    2
    1    2    3    2
    2    4    3    5

matrix3
    3    1
    3    4
    3    1
    5    3

matrix1 + matrix2의 결과
    5    7    7    3
    5    3    5    5
    7    6    6    8

matrix1 - matrix2의 결과
    3   -3   -1   -1
    3   -1   -1    1
    3   -2    0   -2

matrix1 * matrix3의 결과
   32   18
   36   19
   45   25
```

CHAPTER 10

문자열

학습 목표

- C 문자열의 기본적인 개념을 이해합니다.
- C 문자열 관련 라이브러리를 알아봅니다.
- C 문자열의 연산을 실습합니다.
- C++ 문자열의 기본적인 개념을 이해합니다.
- C++ 문자열 관련 라이브러리를 알아봅니다.
- C++ 문자열의 연산을 실습합니다.

03 [Lab] 응용 프로젝트

1 문자열과 관련된 사용자 정의 함수 만들기

이번에는 string 클래스의 멤버 함수를 활용하여 사용자 정의 함수를 4개 만듭니다. 그리고 이를 활용해서 문자열 처리와 관련된 고전적인 문제들을 해결합니다.

라이브러리의 멤버 함수 내부에는 유용한 멤버 함수들이 있습니다. 그리고 필요하다면 이를 활용해서 추가적으로 사용자 정의 함수를 만들어서 활용할 수 있습니다. 클래스의 멤버 함수로 우리가 만든 함수를 끼워 넣을 수는 없지만, 클래스 자료형을 매개변수로 받아서 처리하는 함수는 만들 수 있습니다.

이번 절에서는 C++ 문자열 클래스와 관련된 4개의 함수를 만듭니다. C++ 문자열 클래스에 정의되어 있는 멤버 함수를 활용해서 만들 것입니다. pushFront, pushBack, popFront, popBack이라는 함수를 만들 것인데, 다른 프로그램에서도 쉽게 활용할 수 있게 customized.h라는 헤더 파일에 만듭니다. 이후의 다른 프로그램에서 이를 활용하고자 한다면 customized.h 헤더 파일을 읽어 들여서 사용하면 될 것입니다. 이 함수들이 어떤 처리를 하는지 그림으로 정리해 보면 [그림 10-13]과 같습니다.

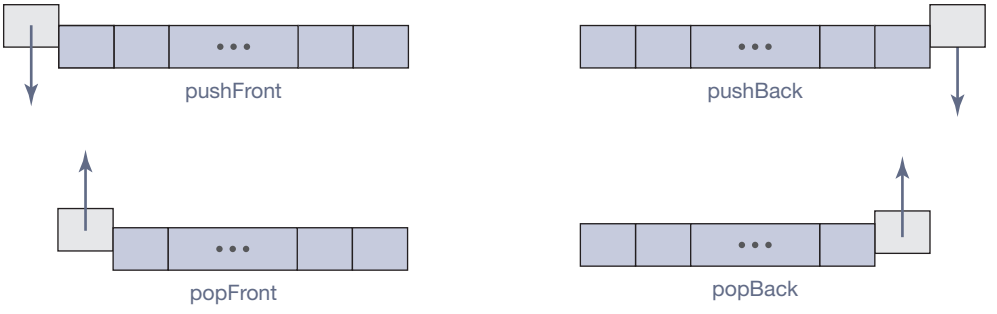

그림 10-13 4가지 사용자 정의 함수의 처리

[프로그램 10-25]는 4개의 사용자 정의 함수를 정리한 헤더 파일입니다. [프로그램 10-26]에서는 간단한 문자열을 사용해서 함수들을 확인합니다.

프로그램 10-25 인터페이스 파일(customized.h) Prg10-25.cpp

```
1  /*************************************************************************
2   * 사용자 정의 함수를 정의하는 헤더 파일                                 *
3   * pushFront 함수는 앞에 문자를 추가                                     *
4   * pushBack 함수는 뒤에 문자를 추가                                      *
5   * popFront 함수는 앞의 문자를 제거                                      *
6   * popBack 함수는 뒤의 문자를 제거                                       *
7   *************************************************************************/
8  #ifndef custom_H
9  #define custom_H
10 #include <iostream>
11 #include <string>
12 using namespace std;
13
14 // pushFront 함수의 정의
15 void pushFront(string& strg, char c)
16 {
17     string temp(1, c);
18     strg.insert(0, temp);
19 }
20 // pushBack 함수의 정의
21 void pushBack(string& strg, char c)
22 {
23     string temp(1, c);
24     strg.append(temp);
25 }
26 // popFront 함수의 정의
27 char popFront(string& strg)
28 {
29     int index = 0;
30     char temp = strg[index];
31     strg.erase(index, 1);
32     return temp;
33 }
34 // popBack 함수의 정의
35 char popBack(string& strg)
36 {
37     int index = strg.size() - 1;
38     char temp = strg[index];
39     strg.erase(index, 1);
40     return temp;
41 }
42 #endif
```

프로그램 10-26 구현 파일(customized.cpp) Prg10-26.cpp

```
/*************************************************************************
 * 이전의 헤더 파일에서 만든                                              *
 * 4개의 사용자 정의 함수를 테스트하는 프로그램                           *
 *************************************************************************/
#include "customized.h"
#include <string>
#include <iostream>
using namespace std;

int main()
{
    // 문자열 선언
    string strg("abcdefgh");
    // pushFront 함수 테스트
    cout << "PushFront 전의 문자열: " << strg << endl;
    pushFront(strg, 'A');
    cout << "pushFront 후의 문자열: " << strg << endl;
    cout << endl;
    // pushBack 함수 테스트
    cout << "pushBack 전의 문자열: " << strg << endl;
    pushBack(strg, 'Z');
    cout << "pushBack 후의 문자열: " << strg << endl;
    cout << endl;
    // popFront 함수 테스트
    cout << "popFront 전의 문자열: " << strg << endl;
    char c1 = popFront(strg);
    cout << "popFront 후의 문자열: " << strg << endl;
    cout << "추출한 문자: " << c1 << endl;
    cout << endl;
    // popBack 함수 테스트
    cout << "popBack 전의 문자열: " << strg << endl;
    char c2 = popBack(strg);
    cout << "popBack 후의 문자열: " << strg << endl;
    cout << "추출한 문자: " << c2 << endl;
    cout << endl;
    return 0;
}
```

실행 결과

```
PushFront 전의 문자열: abcdefgh
pushFront 후의 문자열: Aabcdefgh
```

```
pushBack 전의 문자열: Aabcdefgh
pushBack 후의 문자열: AabcdefghZ

popFront 전의 문자열: AabcdefghZ
popFront 후의 문자열: abcdefghZ
추출한 문자: A

popBack 전의 문자열: abcdefghZ
popBack 후의 문자열: abcdefgh
추출한 문자: Z
```

2 진법 변환 시스템 만들기

컴퓨터 과학에서는 여러 진법을 사용해서 숫자를 표현합니다. 진법과 관련된 자세한 내용은 부록B에서 설명합니다. 진법을 이해하려면 기호symbol와 밑수base를 알아야 합니다. 밑수는 진법 내부에서 사용되는 기호의 수를 나타냅니다. [표 10-2]는 프로그래밍에서 사용되는 밑수에 따른 기호를 정리한 것입니다. 이때 A, B, C, D, E, F는 10, 11, 12, 13, 14, 15를 나타내기 위해서 사용하는 것입니다.

표 10-2 다양한 진법

진법	밑수	기호
binary	2	0, 1
octal	8	0, 1, 2, 3, 4, 5, 6, 7
decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

이러한 기호도 모두 숫자를 표현하기 위한 것이지만, C++ 내부에서 정수로 사용할 수 있는 수는 10진수 뿐입니다. 다른 진법의 숫자는 문자열로 표현해야 합니다.

따라서 10진수에서 다른 진법으로 변환하거나 반대의 경우에는 자료형을 변경해야 합니다. 이번 절에서는 이러한 진법 변환 시스템의 일반적인 설계를 살펴봅니다.

다른 진법을 10진법으로 변환

[그림 10-14]는 길이가 3인 문자열로 표현된 숫자를 10진법의 정수로 변환하는 방법을 나타냈습니다.

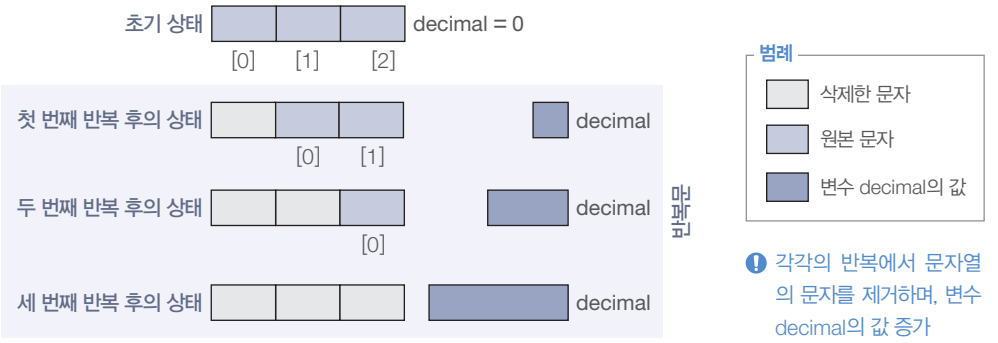

그림 10-14 다른 진법을 10진법으로 변환하기

문자열의 크기만큼 반복하는 반복문을 사용합니다. 반복문은 문자열이 빈 문자열로 될 때 종료할 것입니다. 반복문을 시작하기 전에는 변수 decimal의 값을 0으로 할당합니다. 각각의 반복에서 이전 반복의 decimal 값에 10을 곱합니다. 그리고 문자열의 가장 앞 문자를 삭제하고, 10진수로 변환한 뒤, 이를 decimal에 더해줍니다. 각각의 반복에서 문자열의 문자는 하나씩 제거되지만, 변수 decimal의 값을 증가시킵니다. 반복문이 종료될 때 10진수로 변환이 완료됩니다. 이를 정리해보면 다음과 같습니다.

```
set base                                // 밑수(base)는 2, 8, 16 등으로 할당
decimal = 0
input string
while(string not empty)
{
    decimal *= base;
    ch = popFront(string)               // 이전에 만들었던 popFront 함수
    decimal += findValue(ch)            // 문자를 10진수 값으로 변환
}
output decimal
```

[프로그램 10-27]은 2진수 문자열을 10진수 정수로 바꾸는 예입니다. 이전에 만들었던 popFront 함수를 사용했습니다. 문자열에서 추출한 각각의 값을 10진수 정수로 변환하기 위해서, findValue라는 함수를 만들어서 사용했습니다.

프로그램 10-27 2진수 문자열을 10진수 정수로 바꾸기(cutomized.cpp) Prg10-27.cpp

```
1  /*************************************************************************
2   * 2진수 문자열을 10진수 숫자로 변환하는 프로그램                        *
3   *************************************************************************/
4  #include "customized.h"
```

```
#include <string>
#include <iostream>
using namespace std;
/*************************************************************************
 * 문자를 해당하는 정수로                                                *
 * 변환하는 함수                                                         *
 *************************************************************************/
int findValue(char ch)
{
    return static_cast<int>(ch) - 48;
}

int main()
{
    // 선언, 입력, 유효성 검사
    string binary;
    do
    {
        cout << "2진수 숫자를 입력하세요: ";
        getline(cin, binary);
    } while(binary.find_first_not_of("01") < binary.size());
    // 10진수 숫자로 변환하고 출력
    int base = 2;
    int decimal = 0;
    while(!binary.empty())
    {
        decimal *= base;
        char ch = popFront(binary);
        decimal += findValue(ch);
    }
    cout << "10진수 값: " << decimal;
    return 0;
}
```

실행 결과

```
2진수 숫자를 입력하세요: 11 101
2진수 숫자를 입력하세요: 11101
10진수 값: 29
```

실행 결과

```
2진수 숫자를 입력하세요: 1181
2진수 숫자를 입력하세요: 11111
10진수 값: 31
```

실행 결과

```
2진수 숫자를 입력하세요: 111000111
10진수 값: 455
```

[프로그램 10-27]에서 중요하게 짚고 넘어가야 하는 부분이 3가지 있습니다. 첫 번째는 21~25행입니다. 2진법은 0과 1만으로 구성됩니다. 따라서 유효성 검사로 사용자가 입력한 문자열이 0과 1로만 구성되어 있는지 확인합니다. 이때 이번 장의 앞부분에서 배웠던 find_first_not_of 함수를 활용합니다. 이 함수를 호출해서 0과 1이 아닌 문자가 있을 경우, 인덱스를 리턴하게 했습니다. 만약 리턴된 인덱스가 문자열의 크기보다 작다면, 0과 1이 아닌 문자가 있다는 것이므로, 다시 입력을 받게 했습니다.

두 번째는 31행입니다. 반복문에 들어갈 때마다 변수 decimal에 base(밑수)를 곱합니다. 세 번째는 popFront 함수입니다. 반복을 돌 때마다 문자열에서 가장 앞의 문자 하나를 추출해야 합니다. 이를 위해 customized.h 헤더 파일에 정의한 popFront 함수를 사용했습니다.

10진법을 다른 진법으로 변환

[그림 10-15]는 10진수를 문자가 3개인 문자열로 변환하는 과정을 나타낸 것입니다.

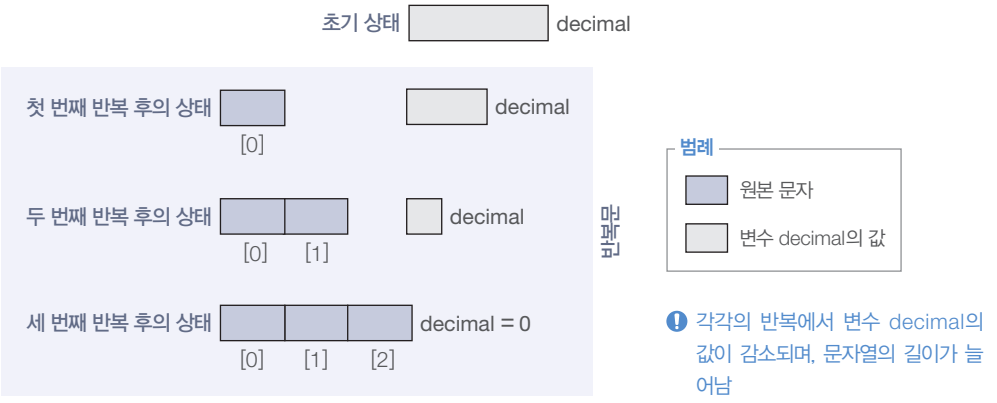

그림 10-15 10진법으로 다른 진법으로 변환하기

10진수 값을 기반으로 반복하는 반복문을 사용합니다. 값이 0이라면 중지합니다. 반복문을 시작하기 전에는 문자열이 비어 있습니다. 각각의 반복에서 이전 반복의 decimal 값을 밑수로 나누고, 나머지도 구합니다. 그리고 나머지를 문자열 앞에 삽입합니다. 각각의 반복에서 변수 decimal의 값은 줄어들지만, 문자열의 크기는 점점 증가합니다. 반복문이 종료될 때 변환이 완료됩니다. 이를 정리해보면 다음과 같습니다.

```
set base                                    // 밑수(base)는 2, 8, 16 등으로 할당
input decimal
while(decimal > 0)
{
    value = decimal % base;
    ch = findChar(value)                    // 숫자를 문자로 변환
    string.pushFront(ch)                    // 이전에 만들었던 pushFront 함수
}
output string
```

[프로그램 10-28]은 10진수 정수를 2진수 문자열로 바꾸는 프로그램입니다.

프로그램 10-28 10진수 정수를 2진수 문자열로 바꾸기(customized.cpp) Prg10-28.cpp

```
1  /*************************************************************************
2   * 10진수 숫자를 2진수 문자열로 변환하는 프로그램                        *
3   *************************************************************************/
4  #include <string>
5  #include "customized.h"
6  #include <iostream>
7  using namespace std;
8
9  /*************************************************************************
10  * 정수를 문자로 변환하는 함수                                           *
11  *************************************************************************/
12 char findChar(int digit)
13 {
14     return char(digit + 48);
15 }
16
17 int main()
18 {
19     // 변수 선언
20     int decimal;
21     int base = 2;
22     string strg;
23     // 10진수 입력 유효성 검사
24     do
25     {
26         cout << "10진수 정수를 입력하세요: ";
27         cin >> decimal;
28     } while(decimal <= 0);
```

```
29      // 2진수로 변환
30      while(decimal > 0)
31      {
32          int digit = decimal % base;
33          char ch = findChar(digit);
34          pushFront(strg, ch);
35          decimal /= base;
36      }
37      // 2진수 출력
38      cout << "2진수: " << strg;
39      return 0;
40  }
```

실행 결과

```
10진수 정수를 입력하세요: 35
2진수: 100011
```

실행 결과

```
10진수 정수를 입력하세요: 7
2진수: 111
```

실행 결과

```
10진수 정수를 입력하세요: 126
2진수: 1111110
```

10진수의 유효성 검사는 do-while 반복문으로 처리했습니다. 2진수를 나타내는 문자열에 문자를 삽입할 때는 customized.h에 정의한 pushFront 함수를 사용했습니다.

CHAPTER 11

클래스 간의 관계

학습 목표

- 클래스 간의 상속을 알아봅니다.
- public과 private의 상관관계를 이해합니다.
- 오버로드와 오버라이드를 구분하여 이해합니다.
- 상속의 접근과 막기를 알아봅니다.
- 상속과 유사한 개념인 연관, 의존을 알아봅니다.

04 [Lab] 응용 프로젝트

1 Tokenizer 클래스 설계하기

구분자로 문자를 사용해서 문자열을 자르는 토큰화는 다양한 곳에 활용됩니다. 예를 들어서 텍스트에서 단어를 추출하는 경우를 생각해봅시다. 텍스트에서 단어를 추출할 때는 공백과 줄바꿈 문자로 자르고 추출하면 될 것입니다. 이때 추출되는 단어가 토큰token이고, 단어를 구분하는 공백과 줄바꿈 문자가 구분 문자delimiter입니다. 예를 들어서 다음과 같은 텍스트는 7개의 토큰(단어)으로 잘립니다.

```
"This is a book about C++ language"
```

C++에는 문자열을 토큰화하는 클래스가 따로 없습니다. 따라서 10장에서 살펴보았던 문자열 라이브러리의 함수들을 활용해서 우리가 직접 Tokenizer 클래스를 만들어봅시다.

클래스 다이어그램

Tokenizer 클래스의 코드를 작성하기 전에 UML 다이어그램으로 클래스들의 관계를 살펴봅시다. 이번 절의 예제에서 사용되는 클래스는 Tokenizer 클래스와 string 클래스 2개입니다. [그림 11-24]는 두 클래스의 관계를 나타낸 것입니다.

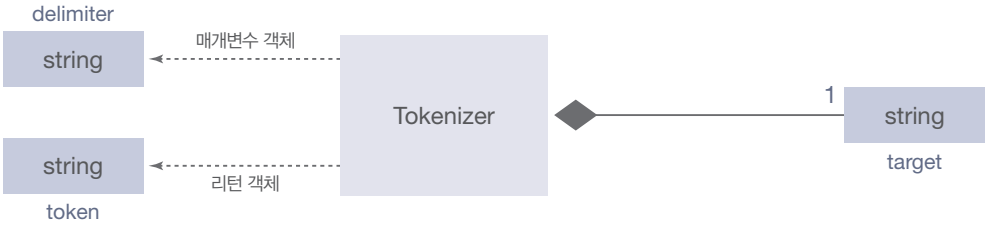

그림 11-24 클래스들의 관계

[그림 11-24]를 보면 Tokenizer 클래스가 문자열 클래스를 3개 사용한다는 것을 알 수 있습니다. target 문자열(텍스트)을 기반으로 구분 문자delimiter로 토큰token을 만드는 것입니다.

이때 target과 delimiter는 하나씩, token은 여러 개가 나옵니다. 이때 Tokenizer 객체와 target, delimiter는 구성 관계입니다. 그리고 Tokenizer 객체와 token은 종속 관계입니다.

시퀀스 다이어그램

[그림 11-25]는 객체들의 시퀀스 다이어그램입니다. target과 tokenizer 객체를 인스턴스화하고 반복문을 사용해서 target에서 토큰을 추출합니다. 사용자는 tokenizer 객체에서 nextToken 함수를 호출합니다. 그리고 함수 내부에서는 target 객체의 find_first_not_of와 find_first_of 함수를 호출합니다. 이를 기반으로 토큰 추출을 완료하면 이를 main 함수로 리턴합니다. 참고로 내부적으로 토큰을 token 객체로 만들어서 활용하겠지만, token 객체는 리턴값으로만 활용되는 종속 관계이므로 시퀀스 다이어그램에는 표시하지 않습니다.

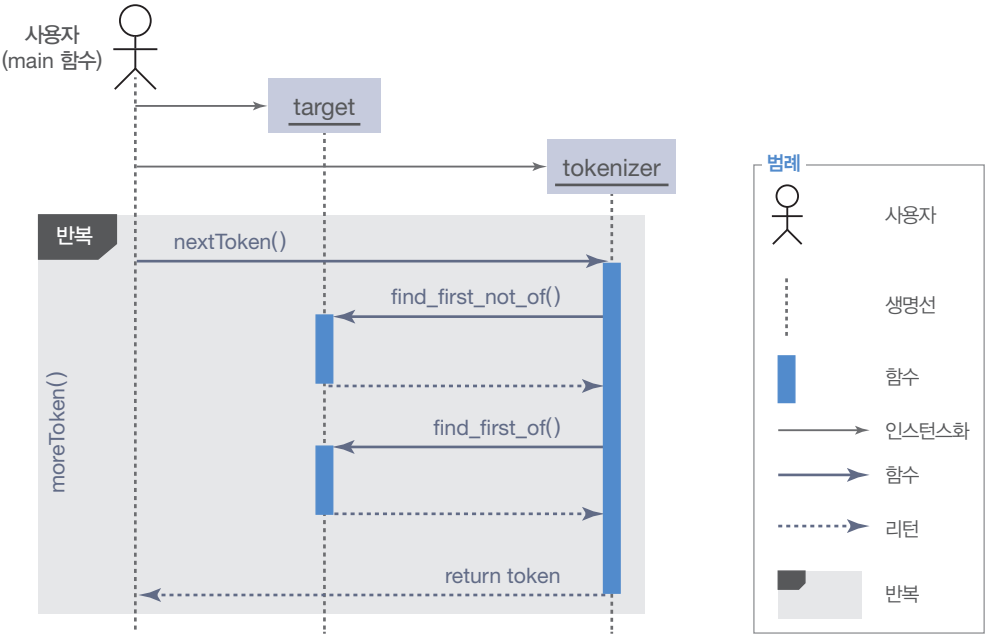

그림 11-25 Tokenizer 설계의 시퀀스 다이어그램

코드 작성

문자열 클래스는 라이브러리가 제공하는 클래스이므로, Tokenizer 클래스만 만들면 됩니다. [프로그램 11-29]는 Tokenizer 클래스의 인터페이스 파일입니다. 데이터 멤버로 문자열 자료형의 target과 delim이 있습니다. 이 클래스는 생성자와 소멸자 이외에 2개의 멤버 함수가 있습니다. 첫 번째는 문자열에 토큰이 있는지 확인하는 멤버 함수이고, 두 번째는 문자열에서 토큰을 추출하는 멤버 함수입니다.

프로그램 11-29 인터페이스 파일(tokenizer.h) Prg11-29.cpp

```
1  /*************************************************************************
2   * Tokenizer 클래스의 인터페이스 파일                                    *
3   *************************************************************************/
4  #ifndef TOKENIZER_H
5  #define TOKENIZER_H
6  #include <iostream>
7  #include <string>
8  using namespace std;
9
10 class Tokenizer
11 {
12     private:
13         string target;
14         string delim;
15         int begin;
16         int end;
17     public:
18         Tokenizer(const string& target, const string& delim);
19         ~Tokenizer();
20         bool moreToken() const;
21         string nextToken();
22 };
23 #endif
```

[프로그램 11-30]은 Tokenizer 클래스의 구현 파일입니다. 생성자와 소멸자를 포함해서 모두 4개의 멤버 함수를 정의합니다.

프로그램 11-30 구현 파일(tokenizer.cpp) Prg11-30.cpp

```
1  /*************************************************************************
2   * Tokenizer 클래스의 구현 파일                                          *
3   *************************************************************************/
4  #include "tokenizer.h"
5
6  // 생성자
7  Tokenizer::Tokenizer(const string& tar, const string& del)
8  : target(tar), delim(del)
9  {
10     begin = target.find_first_not_of(delim, 0);
11     end = target.find_first_of(delim, begin);
```

```
}
// 소멸자
Tokenizer::~Tokenizer()
{
}
// 추가적인 토큰이 있는지 확인
bool Tokenizer::moreToken() const
{
    return(begin != -1);
}
// 다음 토큰 리턴
string Tokenizer::nextToken()
{
    string token = target.substr(begin, end - begin);
    begin = target.find_first_not_of(delim, end);
    end = target.find_first_of(delim, begin);
    return token;
}
```

마지막으로 [프로그램 11-31]은 Tokenizer 클래스를 인스턴스화하여 사용하는 애플리케이션 파일입니다.

프로그램 11-31 애플리케이션 파일(app.cpp) Prg11-31.cpp

```
/*************************************************************************
 * Tokenizer 클래스를 테스트하는 애플리케이션 파일                        *
 *************************************************************************/
#include "tokenizer.h"

int main()
{
    // 토큰화할 문자열
    string target("This is the string to be tokenized. \n");
    // 구분 문자로 사용할 문자열
    string delimit(" \n");          // 띄어쓰기와 줄바꿈을 구분 문자로 사용
    // Tokenizer 객체 인스턴스화
    Tokenizer tokenizer(target, delimit);
    // target 문자열을 기반으로 토큰 찾아 출력
    while(tokenizer.moreToken())
    {
        cout << tokenizer.nextToken() << endl;
```

```
18      }
19      return 0;
20  }
```

실행 결과

```
This
is
the
string
to
be
tokenized.
```

2 대학교 수강 과목 프로그램 만들기

이번 절에서는 대학교에서 활용할 수 있는 수강 과목 등록 프로그램을 만들어봅시다. 연관 관계와 의존 관계를 모두 살펴볼 수 있는 예제입니다.

클래스 다이어그램

[그림 11-26]과 같이 6개의 클래스를 사용합니다.

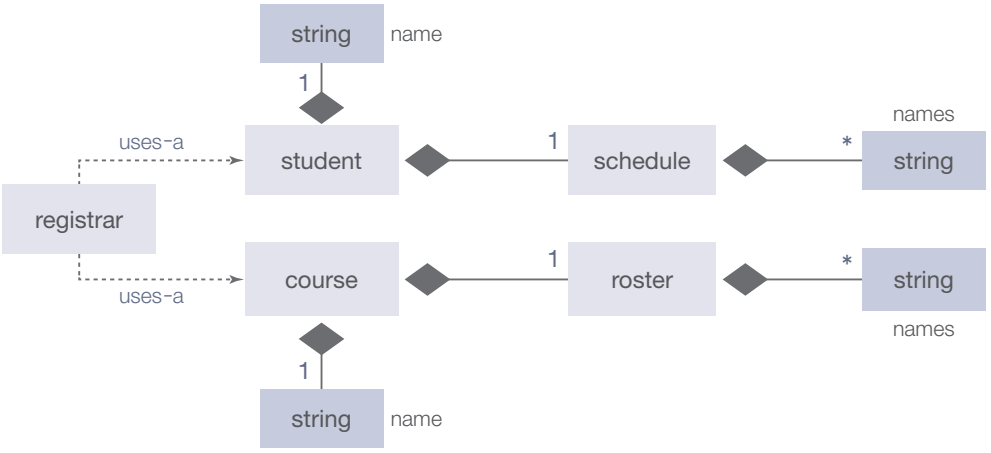

그림 11-26 이번 예제의 프로그램에서 사용하는 클래스

Student 객체는 문자열 객체(학생의 이름)와 Schedule 객체를 포함합니다. Course 객체는 문자열 객체(과목의 이름)와 Roster 객체(명단)를 포함합니다. Schedule 객체와 Roster 객체는 문자열 객체를 포함합니다. 최종적으로 Registrar(등록) 객체에서는 Student 객체와 Course 객체를 사용합니다.

시퀀스 다이어그램

[그림 11-27]을 참고하여, 5개의 사용자 정의 클래스를 만들기 전에 시퀀스 다이어그램을 통해서 클래스들의 상호 작용을 살펴봅시다. Course 클래스에서 Roster 객체를 만들고, Student 객체에서 Schedule 객체를 만듭니다. Registrar, Course, Student 객체는 main 함수에서 만듭니다. 다이어그램은 간단하게 Course 객체와 Student 객체를 하나씩만 만들었습니다.

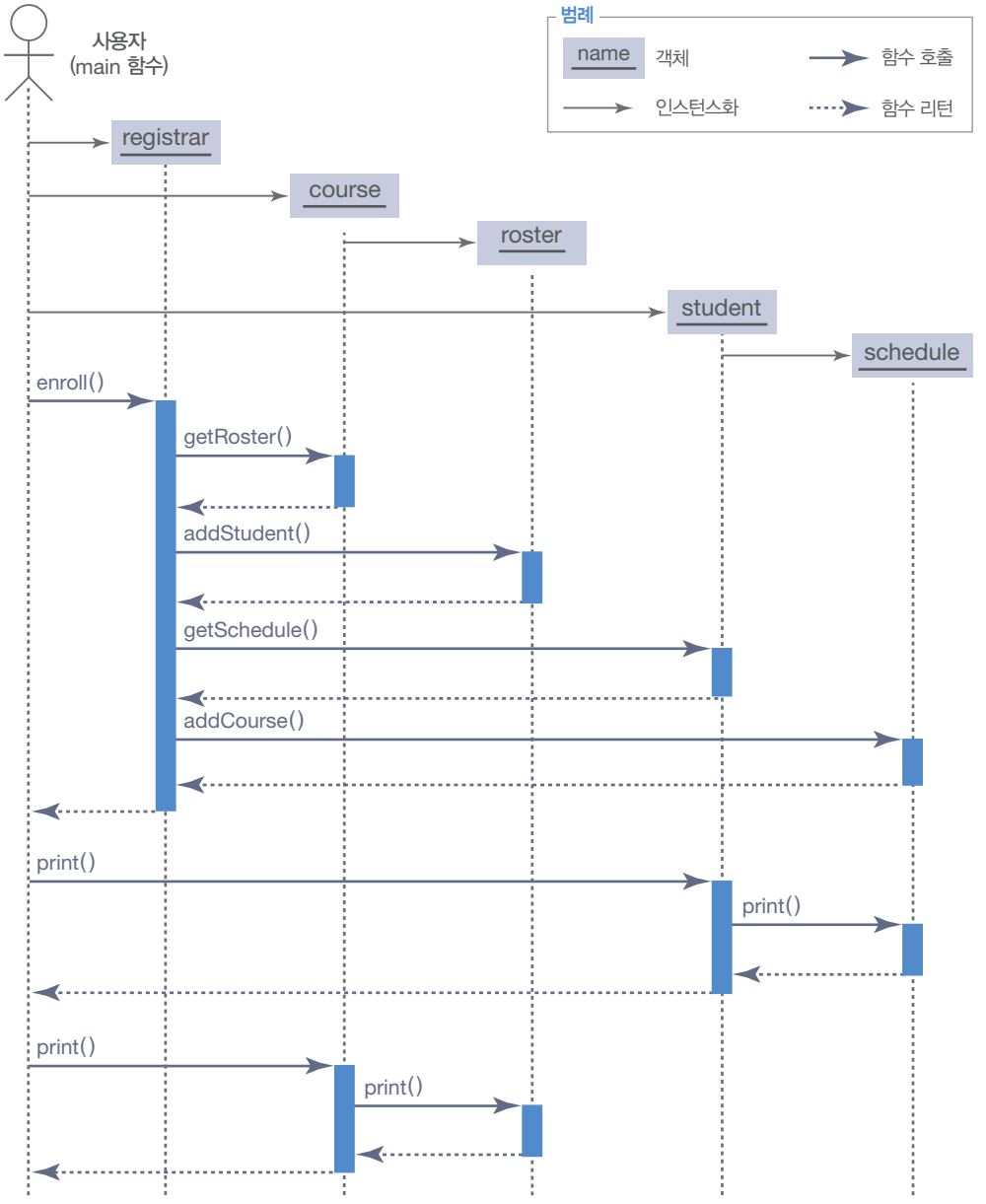

그림 11-27 registrar 프로젝트의 시퀀스 다이어그램

코드 작성

[프로그램 11-32]는 CourseRoster 클래스의 인터페이스 파일이며, [프로그램 11-33]은 구현 파일입니다.

프로그램 11-32 인터페이스 파일(courseRoster.h) Prg11-32.cpp

```
1  /*****************************************************************************
2   * CourseRoster 클래스의 인터페이스 파일                                      *
3   *****************************************************************************/
4  #ifndef COURSEROSTER_H
5  #define COURSEROSTER_H
6  #include <string>
7  #include <iostream>
8  #include <cassert>
9  using namespace std;
10
11 // 클래스 정의
12 class CourseRoster
13 {
14     private:
15         int size;
16         string* stdNames;
17     public:
18         CourseRoster();
19         ~CourseRoster();
20         void addStudent(string studentName);
21         void print() const;
22 };
23 #endif
```

프로그램 11-33 구현 파일(courseRoster.cpp) Prg11-33.cpp

```
1  /*****************************************************************************
2   * CourseRoster 클래스의 구현 파일                                            *
3   *****************************************************************************/
4  #include "courseRoster.h"
5
6  // 생성자
7  CourseRoster::CourseRoster()
8  :size(0)
9  {
10     stdNames = new string[20];
```

```
11 }
12 // 소멸자
13 CourseRoster::~CourseRoster()
14 {
15     delete[] stdNames;
16 }
17 // addStudent 함수의 정의
18 void CourseRoster::addStudent(string studentName)
19 {
20     stdNames[size] = studentName;
21     size++;
22 }
23 // print 함수의 정의
24 void CourseRoster::print() const
25 {
26     cout << "수강하는 학생 목록" << endl;
27     for(int i = 0; i < size; i++)
28     {
29         cout << stdNames[i] << endl;
30     }
31     cout << endl;
32 }
```

[프로그램 11-34]는 Course 클래스의 인터페이스 파일이며, [프로그램 11-35]는 구현 파일입니다.

프로그램 11-34 인터페이스 파일(course.h) Prg11-34.cpp

```
1  /*****************************************************************************
2   * Course 클래스의 인터페이스 파일                                           *
3   *****************************************************************************/
4  #ifndef COURSE_H
5  #define COURSE_H
6  #include <cassert>
7  #include <string>
8  #include <iostream>
9  #include "courseRoster.h"
10 using namespace std;
11
12 // 클래스 정의
13 class Course
14 {
```

```
15      private:
16          string name;
17          int units;
18          CourseRoster* roster;
19      public:
20          Course(string name, int units);
21          ~Course();
22          string getName() const;
23          CourseRoster* getRoster() const;
24          void addStudent(string name);
25          void print() const;
26  };
27  #endif
```

프로그램 11-35 구현 파일(course.cpp) Prg11-35.cpp

```
1   /*************************************************************************
2    * Course 클래스의 구현 파일                                             *
3    *************************************************************************/
4   #include "course.h"
5
6   // 생성자
7   Course::Course(string nm, int ut)
8   : name(nm), units(ut)
9   {
10      roster = new CourseRoster;
11  }
12  // 소멸자
13  Course::~Course()
14  {
15  }
16  // getName 함수의 정의
17  string Course::getName() const
18  {
19      return name;
20  }
21  // addStudent 함수의 정의
22  void Course::addStudent(string name)
23  {
24      roster->addStudent(name);
25  }
26  // getRoster 함수의 정의
```

```
27 CourseRoster* Course::getRoster() const
28 {
29     return roster;
30 }
31 // print 함수의 정의
32 void Course::print() const
33 {
34     cout << "코스 이름: " << name << endl;
35     cout << "과목의 학점: " << units << endl;
36     roster->print();
37 }
```

[프로그램 11-36]은 StudentSchedule 클래스의 인터페이스 파일입니다.

프로그램 11-36 인터페이스 파일(studentSchedule.h) Prg11-36.cpp

```
1  /*****************************************************************************
2   * StudentSchedule 클래스의 인터페이스 파일                                  *
3   *****************************************************************************/
4  #ifndef STUDENTSCHEDULE_H
5  #define STUDENTSCHEDULE_H
6  #include <string>
7  #include <iostream>
8  #include <cassert>
9  using namespace std;
10
11 // 클래스 정의
12 class StudentSchedule
13 {
14     private:
15         int size;
16         string* courseNames;
17     public:
18         StudentSchedule();
19         ~StudentSchedule();
20         void addCourse(string course);
21         void print() const;
22 };
23 #endif
```

[프로그램 11-37]은 구현 파일입니다.

프로그램 11-37 구현 파일(studentSchedule.cpp) Prg11-37.cpp

```
/*************************************************************************
 * StudentSchedule 클래스의 구현 파일                                    *
 *************************************************************************/
#include "studentSchedule.h"

// 생성자
StudentSchedule::StudentSchedule()
:size(0)
{
    courseNames = new string[5];
}
// 소멸자
StudentSchedule::~StudentSchedule()
{
    delete[] courseNames;
}
// addCourse 함수의 정의
void StudentSchedule::addCourse(string name)
{
    courseNames[size] = name;
    size++;
}
// print 함수의 정의
void StudentSchedule::print() const
{
    cout << "수강 과목 목록" << endl;
    for(int i = 0; i < size; i++)
    {
        cout << courseNames[i] << endl;
    }
    cout << endl;
}
```

[프로그램 11-38]은 Student 클래스의 인터페이스 파일입니다.

프로그램 11-38 인터페이스 파일(student.h) Prg11-38.cpp

```
/*************************************************************************
 * Student 클래스의 인터페이스 파일                                      *
 *************************************************************************/
#ifndef STUDENT_H
```

```
#define STUDENT_H
#include <cassert>
#include <string>
#include <iostream>
#include "studentSchedule.h"
using namespace std;

// 클래스 정의
class Student
{
    private:
        string name;
        StudentSchedule* schedule;
    public:
        Student(string name);
        ~Student();
        string getName() const;
        StudentSchedule* getSchedule() const;
        void addCourse(string name);
        void print() const;
};
#endif
```

[프로그램 11-39]는 구현 파일입니다.

프로그램 11-39 구현 파일(student.cpp) Prg11-39.cpp

```
/*************************************************************************
 * Student 클래스의 구현 파일                                            *
 *************************************************************************/
#include "student.h"

// 생성자
Student::Student(string nm)
:name(nm)
{
    schedule = new StudentSchedule;
}
// 소멸자
Student::~Student()
{
}
```

```
16 // getName 함수의 정의
17 string Student::getName() const
18 {
19     return name;
20 }
21 // getSchedule 함수의 정의
22 StudentSchedule* Student::getSchedule() const
23 {
24     return schedule;
25 }
26 // addCourse 함수의 정의
27 void Student::addCourse(string name)
28 {
29     schedule->addCourse(name);
30 }
31 // print 함수의 정의
32 void Student::print() const
33 {
34     cout << "학생 이름: " << name << endl;
35     schedule->print();
36 }
```

[프로그램 11-40]은 Registrar 클래스의 인터페이스 파일이며, [프로그램 11-41]은 구현 파일입니다.

프로그램 11-40 인터페이스 파일(registrar.h) Prg11-40.cpp

```
1  /*************************************************************************
2   * Registrar 클래스의 인터페이스 파일                                    *
3   *************************************************************************/
4  #ifndef REGISTRAR_H
5  #define REGISTRAR_H
6  #include "course.h"
7  #include "student.h"
8
9  // 클래스 정의
10 class Registrar
11 {
12     public:
13         Registrar();
14         ~Registrar();
```

```
15          void enroll(Student student, Course course);
16  };
17  #endif
```

프로그램 11-41 구현 파일(registrar.cpp) Prg11-41.cpp

```
1   /*************************************************************************
2    * Registrar 클래스의 구현 파일                                          *
3    *************************************************************************/
4   #include "registrar.h"
5
6   // 생성자
7   Registrar::Registrar()
8   {
9   }
10  // 소멸자
11  Registrar::~Registrar()
12  {
13  }
14  // enroll 함수
15  void Registrar::enroll(Student student, Course course)
16  {
17      (course.getRoster())->addStudent(student.getName());
18      (student.getSchedule())->addCourse(course.getName());
19  }
```

마지막으로 [프로그램 11-42]는 애플리케이션 파일입니다. 애플리케이션 내부에서 registrar 객체는 딱 하나만 만들어진다는 것에 주목하세요. 20장에서 객체를 하나만 만들 수 있게 강제하는 방법을 배울 예정이지만(싱글 톤 패턴), 일단은 객체를 하나만 만들어서 활용하도록 합시다.

[프로그램 11-42]에서는 Student 객체를 3개, Course 객체를 3개 만들고 있습니다. 그리고 registrar 객체를 사용해서, 학생을 특정 코스(과목)에 등록합니다. StudentSchedule 클래스와 CourseRoster 클래스를 직접적으로 인스턴스화하지 않는다는 것도 주목하세요.

프로그램 11-42 애플리케이션 파일(app.cpp) Prg11-42.cpp

```
1   /*************************************************************************
2    * 모든 클래스를 사용하는 애플리케이션 파일                              *
3    *************************************************************************/
4   #include "registrar.h"
```

```
5
6   int main()
7   {
8       // Registrar 객체 인스턴스화
9       Registrar registrar;
10      // Student 객체 인스턴스화
11      Student student1("John");
12      Student student2("Mary");
13      Student student3("Ann");
14      // Course 객체 3개 인스턴스화
15      Course course1("CIS101", 4);
16      Course course2("CIS102", 3);
17      Course course3("CIS103", 3);
18      // Registrar 객체를 기반으로 학생이 과목 수강 등록
19      registrar.enroll(student1, course1);
20      registrar.enroll(student1, course2);
21      registrar.enroll(student2, course1);
22      registrar.enroll(student2, course3);
23      registrar.enroll(student3, course1);
24      // Student 객체의 정보 출력
25      student1.print();
26      student2.print();
27      student3.print();
28      // Course 객체의 정보 출력
29      course1.print();
30      course2.print();
31      course3.print();
32      return 0;
33  }
```

실행 결과

```
학생 이름: John
수강 과목 목록
CIS101
CIS102

학생 이름: Mary
수강 과목 목록
CIS101
CIS103

학생 이름: Ann
```

```
수강 과목 목록
CIS101

코스 이름: CIS101
과목의 학점: 4
수강하는 학생 목록
John
Mary
Ann

코스 이름: CIS102
과목의 학점: 3
수강하는 학생 목록
John

코스 이름: CIS103
과목의 학점: 3
수강하는 학생 목록
Mary
```

CHAPTER 13

연산자 오버로드

학습 목표

- 연산자 오버로드의 역할을 이해합니다.
- 연산자 함수의 역할을 이해합니다.
- 멤버 함수로 오버로드하는 것이 적합한 연산자를 살펴보겠습니다.
- 비멤버 함수로 오버로드하는 방법을 학습합니다.
- 자료형 변환이 일어나는 과정을 이해합니다.

06 [Lab] 응용 프로젝트

이번 절에서는 연산자 오버로드를 활용하는 클래스를 3개 만들어봅시다. 각각 분수를 나타내는 Fraction 클래스, 날짜를 나타내는 Date 클래스, 다항식을 나타내는 Poly 클래스입니다.

1 연산자 오버로드를 활용해 Fraction 클래스 확장하기

7장에서 분수를 나타내는 Fraction 클래스를 만들어보았습니다. 이번에는 Fraction 클래스에 연산자 오버로드를 적용해서 확장해봅시다.

인터페이스 파일

[프로그램 13-10]은 인터페이스 파일입니다.

프로그램 13-10 인터페이스 파일(fraction.h) Prg13-10.cpp

```
1  /*************************************************************************
2   * Fraction 클래스의 인터페이스 파일                                      *
3   *************************************************************************/
4  #ifndef Fraction_H
5  #define Fraction_H
6  #include <iostream>
7  #include <cassert>
8  #include <iomanip>
9  #include <cmath>
10 using namespace std;
11
12 // Fraction 클래스
13 class Fraction
14 {
15     private:
16         int numer;
17         int denom;
18         int gcd(int n, int m = 1);          // 헬퍼 함수
19         void normalize();                   // 헬퍼 함수
```

```
20      public:
21          Fraction(int numer, int denom);       // 매개변수가 있는 생성자
22          Fraction(double value);               // 매개변수가 있는 생성자
23          Fraction();                           // 기본 생성자
24          Fraction(const Fraction& fract);      // 복사 생성자
25          ~Fraction();                          // 소멸자
26          // 멤버 연산자
27          operator double();                                  // 변환
28          const Fraction operator+() const;                   // 양수
29          const Fraction operator-() const;                   // 음수
30          Fraction& operator++();                             // 전위 증가
31          Fraction& operator--();                             // 전위 감소
32          const Fraction operator++(int);                     // 후위 증가
33          const Fraction operator--(int);                     // 후위 감소
34          Fraction& operator=(const Fraction& right);     // 할당 연산자
35          Fraction& operator+=(const Fraction& right);    // 복합 할당 연산자
36          Fraction& operator-=(const Fraction& right);    // 복합 할당 연산자
37          Fraction& operator*=(const Fraction& right);    // 복합 할당 연산자
38          Fraction& operator/=(const Fraction& right);    // 복합 할당 연산자
39          // friend 수학 연산자
40          friend const Fraction operator+
41                 (const Fraction& left, const Fraction& right);
42          friend const Fraction operator-
43                 (const Fraction& left, const Fraction& right);
44          friend const Fraction operator*
45                 (const Fraction& left, const Fraction& right);
46          friend const Fraction operator/
47                 (const Fraction& left, const Fraction& right);
48          // friend 관계 연산자
49          friend bool operator==
50                 (const Fraction& left, const Fraction& right);
51          friend bool operator!=
52                 (const Fraction& left, const Fraction& right);
53          friend bool operator<
54                 (const Fraction& left, const Fraction& right);
55          friend bool operator<=
56                 (const Fraction& left, const Fraction& right);
57          friend bool operator>
58                 (const Fraction& left, const Fraction& right);
59          friend bool operator>=
60                 (const Fraction& left, const Fraction& right);
61          // 추출 연산자와 삽입 연산자
```

```
62          friend istream& operator>>(istream& left, Fraction& right);
63          friend ostream& operator<<(ostream& left, const Fraction& right);
64  };
65  #endif
```

구현 파일

[프로그램 13-11]은 Fraction 클래스의 구현 파일입니다.

프로그램 13-11 구현 파일(fraction.cpp) Prg13-11.cpp

```
1   /*************************************************************************
2    * Fraction 클래스의 구현 파일                                           *
3    *************************************************************************/
4   #include "fraction.h"
5
6   // 매개변수가 있는 생성자
7   Fraction::Fraction(int num, int den = 1)
8   : numer(num), denom(den)
9   {
10      normalize();
11  }
12  // 매개변수가 있는 생성자
13  Fraction::Fraction(double value)
14  {
15      denom = 1;
16      while((value - static_cast<int>(value)) > 0.0)
17      {
18          value *= 10.0;
19          denom *= 10;
20      }
21      numer = static_cast<int>(value);
22      normalize();
23  }
24  // 기본 생성자
25  Fraction::Fraction()
26  : numer(0), denom(1)
27  {
28  }
29  // 복사 생성자
30  Fraction::Fraction(const Fraction& fract)
31  : numer(fract.numer), denom(fract.denom)
```

```
{
}
// 소멸자
Fraction::~Fraction()
{
}
// 변환 연산자
Fraction::operator double()
{
    double num = static_cast<double>(numer);
    return(num / denom);
}
// 양수 연산자
const Fraction Fraction::operator+() const
{
    Fraction temp(+numer, denom);
    return temp;
}
// 음수 연산자
const Fraction Fraction::operator-() const
{
    Fraction temp(-numer, denom);
    return temp;
}
// 전위 증가 연산자
Fraction& Fraction::operator++()
{
    numer = numer + denom;
    this->normalize();
    return *this;
}
// 전위 감소 연산자
Fraction& Fraction::operator--()
{
    numer = numer - denom;
    this->normalize();
    return *this;
}
// 후위 증가 연산자
const Fraction Fraction::operator++(int)
{
    Fraction temp(numer, denom);
```

```
74      ++(*this);
75      return temp;
76  }
77  // 후위 감소 연산자
78  const Fraction Fraction::operator--(int)
79  {
80      Fraction temp(numer, denom);
81      --(*this);
82      return temp;
83  }
84  // 할당 연산자
85  Fraction& Fraction::operator=(const Fraction& right)
86  {
87      if(*this != right)
88      {
89          numer = right.numer;
90          denom = right.denom;
91      }
92      return *this;
93  }
94  // 복합 할당 연산자(+=)
95  Fraction& Fraction::operator+=(const Fraction& right)
96  {
97      numer  = numer * right.denom + denom * right.numer;
98      denom = denom * right.denom;
99      normalize();
100     return *this;
101 }
102 // 복합 할당 연산자(-=)
103 Fraction& Fraction::operator-=(const Fraction& right)
104 {
105     numer = numer * right.denom - denom * right.numer;
106     denom = denom * right.denom;
107     normalize();
108     return *this;
109 }
110 // 복합 할당 연산자(*=)
111 Fraction& Fraction::operator*=(const Fraction& right)
112 {
113     numer = numer * right.numer;
114     denom = denom * right.denom;
115     normalize();
```

```
116     return *this;
117 }
118 // 복합 할당 연산자(/=)
119 Fraction& Fraction::operator/=(const Fraction& right)
120 {
121     numer = numer * right.denom;
122     denom = denom * right.numer;
123     normalize();
124     return *this;
125 }
126 // friend 덧셈 연산자
127 const Fraction operator+(const Fraction& left, const Fraction& right)
128 {
129     int newNumer = left.numer * right.denom + right.numer * left.denom;
130     int newDenom = left.denom * right.denom;
131     Fraction result(newNumer, newDenom);
132     return result;
133 }
134 // friend 뺄셈 연산자
135 const Fraction operator-(const Fraction& left, const Fraction& right)
136 {
137     int newNumer = left.numer * right.denom - right.numer * left.denom;
138     int newDenom = left.denom * right.denom;
139     Fraction result(newNumer, newDenom);
140     return result;
141 }
142 // friend 곱셈 연산자
143 const Fraction operator*(const Fraction& left, const Fraction& right)
144 {
145     int newNumer = left.numer * right.numer;
146     int newDenom = left.denom * right.denom;
147     Fraction result(newNumer, newDenom);
148     return result;
149 }
150 // friend 나눗셈 연산자
151 const Fraction operator/(const Fraction& left, const Fraction& right)
152 {
153     int newNumer = left.numer * right.denom;
154     int newDenom = left.denom * right.numer;
155     Fraction result(newNumer, newDenom);
156     return result;
157 }
```

```
158 // friend == 연산자
159 bool operator==(const Fraction& left, const Fraction& right)
160 {
161     return(left.numer * right.denom == right.numer * left.denom);
162 }
163 // friend != 연산자
164 bool operator!=(const Fraction& left, const Fraction& right)
165 {
166     return(left.numer * right.denom != right.numer * left.denom);
167 }
168 // friend < 연산자
169 bool operator<(const Fraction& left, const Fraction& right)
170 {
171     return(left.numer * right.denom < right.numer * left.denom);
172 }
173 // friend <= 연산자
174 bool operator<=(const Fraction& left, const Fraction& right)
175 {
176     return(left.numer * right.denom <= right.numer * left.denom);
177 }
178 // friend > 연산자
179 bool operator>(const Fraction& left, const Fraction& right)
180 {
181     return(left.numer * right.denom > right.numer * left.denom);
182 }
183 // friend >= 연산자
184 bool operator>=(const Fraction& left, const Fraction& right)
185 {
186     return(left.numer * right.denom >= right.numer * left.denom);
187 }
188 // friend 추출 연산자
189 istream& operator >>(istream& left, Fraction& right)
190 {
191     cout << "분자를 입력하세요: ";
192     left >> right.numer;
193     cout << "분모를 입력하세요: ";
194     left >> right.denom;
195     right.normalize();
196     return left;
197 }
198 // friend 삽입 연산자
199 ostream& operator<<(ostream& left, const Fraction& right)
```

```
200 {
201     left << right.numer << "/" << right.denom;
202     return left;
203 }
204 // 헬퍼 함수(최대 공약수)
205 int Fraction::gcd(int n, int m)
206 {
207     int gcd = 1;
208     for(int k = 1; k <= n && k <= m; k++)
209     {
210         if(n % k == 0 && m % k == 0)
211         {
212             gcd = k;
213         }
214     }
215     return gcd;
216 }
217 // 헬퍼 함수(약분)
218 void Fraction::normalize()
219 {
220     if(denom == 0)
221     {
222         cout << "유효하지 않은 분수입니다. 프로그램을 중단합니다." << endl;
223         assert(false);
224     }
225     if(denom < 0)
226     {
227         denom = -denom;
228         numer = -numer;
229     }
230     int divisor = gcd(abs(numer), abs(denom));
231     numer = numer / divisor;
232     denom = denom / divisor;
233 }
```

애플리케이션 파일

[프로그램 13-12]는 Fraction 클래스의 동작을 확인하는 애플리케이션 파일입니다.

프로그램 13-12 애플리케이션 파일(app.cpp) Prg13-12.cpp

```
/*****************************************************************************
 * Fraction 클래스를 사용하는 애플리케이션 파일                              *
 *****************************************************************************/
#include "fraction.h"

int main()
{
    // 객체 생성과 양수 음수 연산자 사용
    Fraction fract1(2, 3);
    Fraction fract2(1, 2);
    cout << "fract1: " << fract1 << endl;
    cout << "fract2: " << fract2 << endl;
    cout << "+fract1: " << +fract1 << endl;
    cout << "-fract2: " << -fract2 << endl << endl;
    // ++ -- 연산자 사용
    Fraction fract3(3, 4);
    Fraction fract4(4, 5);
    Fraction fract5(5, 6);
    Fraction fract6(6, 7);
    cout << "fract3: " << fract3 << endl;
    cout << "fract4: " << fract4 << endl;
    cout << "fract5: " << fract5 << endl;
    cout << "fract6: " << fract6 << endl << endl;
    ++fract3;
    --fract4;
    fract5++;
    fract6--;
    cout << "++fract3: " << fract3 << endl;
    cout << "--fract4: " << fract4 << endl;
    cout << "fract5++: " << fract5 << endl;
    cout << "fract6--: " << fract6 << endl << endl;
    // 복합 할당 연산자 사용
    Fraction fract7(3, 5);
    Fraction fract8(4, 7);
    Fraction fract9(5, 8);
    Fraction fract10(7, 9);
    fract3 += 2;
    fract4 -= 3;
    fract5 *= 4;
    fract6 /= 5;
```

```
41      cout << "fract7 += 2: " << fract7 << endl;
42      cout << "fract8 -= 3: " << fract8 << endl;
43      cout << "fract9 *= 4: " << fract9 << endl;
44      cout << "fract10 /= 5: " << fract10 << endl << endl;
45      // 객체 생성과 friend 연산자 함수 사용
46      Fraction fract11(1, 2);
47      Fraction fract12(3, 4);
48      cout << "fract11: " << fract11 << endl;
49      cout << "fract12: " << fract12 << endl;
50      cout << "fract11 + fract12: " << fract11 + fract12 << endl;
51      cout << "fract11 - fract12: " << fract11 - fract12 << endl;
52      cout << "fract11 * fract12: " << fract11 * fract12 << endl;
53      cout << "fract11 / fract12: " << fract11 / fract12 << endl << endl;
54      // 객체 생성과 관계 연산자 사용
55      Fraction fract13(2, 3);
56      Fraction fract14(1, 3);
57      cout << "fract13: " << fract13 << endl;
58      cout << "fract14: " << fract14 << endl;
59      cout << "fract13 == fract14: " << boolalpha;
60      cout << (fract13 == fract14) << endl;
61      cout << "fract13 != fract14: " << boolalpha;
62      cout << (fract13 != fract14) << endl;
63      cout << "fract13 >  fract14: " << boolalpha;
64      cout << (fract13 >  fract14) << endl;
65      cout << "fract13 <  fract14: " << boolalpha;
66      cout << (fract13 <  fract14) << endl << endl;
67      // 객체 생성과 변환
68      Fraction fract15(5);          // 정수를 변환
69      Fraction fract16(23.45);      // 부동 소수점을 변환
70      cout << "fract15: " << fract15 << endl;
71      cout << "fract16: " << fract16 << endl << endl;
72      // double 자료형으로 변환
73      Fraction fract17(9, 13);
74      cout << "double 자료형으로 변환한 fract17(9, 13): ";
75      cout << setprecision(2) << fract17.operator double() << endl << endl;
76      // 삽입 연산자 사용
77      Fraction fract18;
78      cin >> fract18;
79      cout << "fract18: " << fract18 << endl;
80      return 0;
81  }
```

실행 결과

```
fract1: 2/3
fract2: 1/2
+fract1: 2/3
-fract2: -1/2

fract3: 3/4
fract4: 4/5
fract5: 5/6
fract6: 6/7

++fract3: 7/4
--fract4: -1/5
fract5++: 11/6
fract6--: -1/7

fract7 += 2: 3/5
fract8 -= 3: 4/7
fract9 *= 4: 5/8
fract10 /= 5: 7/9

fract11: 1/2
fract12: 3/4
fract11 + fract12: 5/4
fract11 - fract12: -1/4
fract11 * fract12: 3/8
fract11 / fract12: 2/3

fract13: 2/3
fract14: 1/3
fract13 == fract14: false
fract13 != fract14: true
fract13 > fract14: true
fract13 < fract14: false

fract15: 5/1
fract16: 469/20

double 자료형으로 변환한 fract17(9, 13): 0.69

분자를 입력하세요: 6
분모를 입력하세요: 7
Fract18 : 6/7
```

역자 static_cast<double>(fraction17) 또는 (double)fraction17도 활용할 수 있습니다.

2 연산자 오버로드를 활용해 Date 클래스 만들기

월, 일, 연(예: 2/5/2020)이라는 형태로 정수 3개를 조합해서 날짜를 표현하는 Date 클래스를 만들어봅시다. 연산자 오버로드를 적용해서 날짜에서 특정 날을 더하고 빼거나, 날짜끼리 더하고 빼거나, 비교할 수 있는 기능을 구현합니다.

날짜는 기본적으로 시작 날짜에서 며칠이 지났는지를 나타냅니다. 대부분의 국가에서 일반적으로 사용하는 그레고리력은 1월 1일을 시작일로 합니다. 그런데 단순하게 며칠이 지났는지를 적으면, 오늘날을 표현하는데 80만에 가까운 큰 숫자를 사용해야 하므로 연, 월, 일이라는 개념을 사용합니다. 그런데 월의 날짜 수는 고정되어 있지 않으며, 연도도 윤년 등이 발생합니다. 따라서 구현이 조금 복잡합니다.

전략

그레고리력의 복잡함을 조금 줄일 수 있게 다음과 같은 전략을 사용합니다.

- 16세기에 달력이 조금 변경되었으므로, 이 이전의 날짜까지 구현하는 것은 너무 복잡합니다. 따라서 이를 피하고자 1900년 1월 1일부터 날짜를 세겠습니다.
- 객체 지향 프로그래밍 원칙(다른 데이터 멤버를 기반으로 계산할 수 있는 값은 데이터 멤버로 만들지 않음)에 따를 수 있게 연, 월, 일만 데이터 멤버로 만듭니다. 1900년 1월 1일부터 지난 날 수, 요일 등을 찾을 때는 멤버 함수를 활용합니다.
- findTotalDays라는 시작 날짜로부터 지난 날 수를 구하는 함수를 만들 것입니다. 이때 시작 날짜는 1900년 1월 1일로 합니다. 이 날은 월요일입니다.
- 복합 할당 연산자, 증가 연산자, 감소 연산자를 정의해서 날짜를 더하고 빼는 기능을 구현합니다.
- 증가 연산자와 감소 연산자를 사용하면서 날짜를 다음날과 이전날로 변경합니다. 날짜 변경으로 인해서 연과 월이 바뀔 수 있으므로, plusReset과 minusReset이라는 추가적인 private 멤버 함수를 활용합니다.
- 두 날짜(Date 객체)를 빼면, 날짜 차이를 구할 수 있게 합니다. 각 Date 객체의 findTotalDays 함수를 호출해서 시작 날짜로부터 지난 날 수를 구하고, 이를 빼면 쉽게 날짜 차이를 구할 수 있습니다.

불변 속성

7장에서 언급했던 것처럼 클래스 불변 속성을 생각해줘야 합니다. Date 객체를 구현할 때 주의해야 하는 불변 속성은 다음과 같습니다.

- 월은 1~12 사이의 값을 갖습니다.
- 날짜는 1~해당 월의 날짜(28, 30, 31) 사이의 값을 갖습니다. 만약 윤년이라면 2월의 날은 1~29 사이의 값을 갖습니다.
- 연은 시작 날짜의 연도(1900년)보다 크거나 같아야 합니다.

인터페이스 파일

인터페이스 파일, 구현 파일, 애플리케이션 파일을 분리해서 만듭니다. 인터페이스 파일 date.h는 [프로그램 13-13]과 같습니다.

프로그램 13-13 인터페이스 파일(date.h) Prg13-13.cpp

```
1  /*****************************************************************************
2   * Date 클래스의 인터페이스 파일                                             *
3   *****************************************************************************/
4  #ifndef DATE_H
5  #define DATE_H
6  #include <iostream>
7  #include <cmath>
8  #include <cassert>
9  #include <string>
10 using namespace std;
11
12 class Date
13 {
14     private:
15         // 인스턴스 데이터 멤버
16         int month;
17         int day;
18         int year;
19         // 정적 데이터 멤버와 멤버 함수
20         static const int startWeekDay;
21         static const int startYear;
22         static const int daysInMonths[];
23         static const string daysOfWeek[];
24         static const string monthsOfYear[];
25         static bool isLeap(int year);
26         // private 헬퍼 함수
27         bool isValid() const;
28         string findWeekDay();
29         int findTotalDays() const;
```

```
30          void plusReset();
31          void minusReset();
32      public:
33          // 생성자와 소멸자
34          Date(int month, int day, int year);
35          Date();
36          ~Date();
37          // 연산자 오버로드
38          Date& operator++();
39          Date& operator--();
40          Date operator++(int);
41          Date operator--(int);
42          Date& operator+=(int days);
43          Date& operator-=(int days);
44          bool operator==(const Date& right) const;
45          bool operator!=(const Date& right) const;
46          Date& operator=(const Date& right);
47          // friend 연산자 함수
48          friend int operator-(const Date& date1, const Date& date2);
49          friend ostream& operator<<(ostream& output, const Date& date);
50  };
51  #endif
```

구현 파일

[프로그램 13-14]는 구현 파일입니다. 구현 파일에서는 인터페이스 파일에서 정의했던 모든 멤버 함수들을 정의합니다. 인터페이스 파일에서 선언한 정적 멤버의 초기화도 구현 파일에서 합니다.

프로그램 13-14 구현 파일(date.cpp) Prg13-14.cpp

```
1   /*************************************************************************
2    * Date 클래스의 구현 파일                                               *
3    *************************************************************************/
4   #include "date.h"
5
6   // 매개변수가 있는 생성자
7   Date::Date(int m, int d, int y)
8   : month(m), day(d), year(y)
9   {
10      if(!isValid())
11      {
```

```
12          cout << "유효하지 않은 날짜입니다. 프로그램을 중단합니다." << endl;
13          assert(false);
14      }
15  }
16  // 기본 생성자
17  Date::Date()
18  : month(1), day(1), year(1900)
19  {
20  }
21  // 소멸자
22  Date::~Date()
23  {
24  }
25  // 전위 증가 연산자
26  Date& Date::operator++()
27  {
28      day++;
29      plusReset();
30      return *this;
31  }
32  // 전위 감소 연산자
33  Date& Date::operator--()
34  {
35      day--;
36      minusReset();
37      return *this;
38  }
39  // 후위 증가 연산자
40  Date Date::operator++(int)
41  {
42      Date temp(month, day, year);
43      ++(*this);
44      return temp;
45  }
46  // 후위 감소 연산자
47  Date Date::operator--(int)
48  {
49      Date temp(month, day, year);
50      --(*this);
51      return temp;
52  }
53  // 복합 할당 연산자 +=
```

```
54  Date& Date::operator+=(int days)
55  {
56      for(int i = 1; i <= days; i++)
57      {
58          ++(*this);
59      }
60      return *this;
61  }
62  // 복합 할당 연산자 -=
63  Date& Date::operator-=(int days)
64  {
65      for(int i = days; i >= 1; i--)
66      {
67          --(*this);
68      }
69      return *this;
70  }
71  // == 연산자
72  bool Date::operator==(const Date& right) const
73  {
74      bool fact1 = (month == right.month);
75      bool fact2 = (day == right.day);
76      bool fact3 = (year == right.year);
77      return(fact1 && fact2 && fact3);
78  }
79  // != 연산자
80  bool Date::operator!=(const Date& right) const
81  {
82      return !(*this == right);
83  }
84  // 할당 연산자
85  Date& Date::operator=(const Date& right)
86  {
87      if(*this != right)        // 자기 할당 확인
88      {
89          month = right.month;
90          day = right.day;
91          year = right.year;
92      }
93      return *this;
94  }
95  // friend 뺄셈 연산자
```

```
96  int operator-(const Date& date1, const Date& date2)
97  {
98      return(date1.findTotalDays() - date2.findTotalDays());
99  }
100 // friend << 연산자
101 ostream& operator<<(ostream& output, const Date& date)
102 {
103     output << Date::daysOfWeek[(date.findTotalDays()
104                         + Date::startWeekDay) % 7] << " ";
105     output << Date::monthsOfYear[date.month] << " ";
106     output << date.day << " ";
107     output << date.year << endl;
108     return output;
109 }
110 // 유효성 검사 헬퍼 함수
111 bool Date::isValid() const
112 {
113     bool validMonth = (month >= 1) &&(month <= 12);
114     bool validYear = (year >= startYear);
115     bool validDay = (day >= 1) &&(day <= (Date::daysInMonths[month]
116                     +(isLeap(year) && month == 2)));
117     return(validMonth && validYear && validDay);
118 }
119 // 덧셈 후 정리하는 헬퍼 함수
120 void Date::plusReset()
121 {
122     bool extraDay = (isLeap(year) && month == 2);
123     if(day > daysInMonths[month] + extraDay)
124     {
125         day = 1;
126         month++;
127     }
128     if(month > 12)
129     {
130         month = 1;
131         year++;
132     }
133 }
134 // 뺄셈 후 정리하는 헬퍼 함수
135 void Date::minusReset()
136 {
137     if(day < 1)
```

```
    {
        month--;
        if(month < 1)
        {
            month = 12;
            year--;
        }
        bool extraDay = isLeap(year) && (month == 2);
        day = daysInMonths[month] + extraDay;
    }
}
// 전체 날 수를 구하는 헬퍼 함수
int Date::findTotalDays() const
{
    int totalDays = 0;
    int currentYear = startYear;
    while(year > currentYear)
    {
        totalDays += 365 + isLeap(currentYear);
        currentYear++;
    }
    int currentMonth = 1;
    while(month > currentMonth)
    {
        totalDays += daysInMonths[currentMonth];
        if(currentMonth == 2)
        {
            totalDays += isLeap(year);
        }
        currentMonth++;
    }
    totalDays += day - 1;
    return totalDays;
}
// 정적 데이터 멤버 초기화
const int Date::startWeekDay = 1;
const int Date::startYear = 1900;
const int Date::daysInMonths[] = {0, 31, 28, 31, 30, 31,
                30, 31, 31, 30, 31, 30, 31};
const string Date::daysOfWeek[]= {"Sun", "Mon", "Tue", "Wed",
                "Thr", "Fri", "Sat"};
const string Date::monthsOfYear[] = {"", "Jan", "Feb", "Mar", "Apr",
```

```
180                 "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
181 // 정적 멤버 함수 정의
182 bool Date::isLeap(int year)
183 {
184     return(year % 400 == 00) || ((year % 4 == 0)&&(year % 100 != 0));
185 }
```

애플리케이션 파일

애플리케이션 파일에서는 Date 클래스의 객체를 만들고, 이를 활용합니다. 클래스를 활용할 때는 공용 인터페이스를 참고해야 하고, 컴파일된 구현 파일을 읽어 들여서 사용해야 합니다.

프로그램 13-15 애플리케이션 파일(app.cpp) Prg13-15.cpp

```
1  /*************************************************************
2   * Date 클래스를 사용하는 애플리케이션 파일                  *
3   *************************************************************/
4  #include "date.h"
5
6  int main()
7  {
8      // Date 객체 2개 생성 후 출력
9      Date date1(2, 8, 2014);
10     Date date2(10, 15, 1944);
11     cout << "date1 = " << date1;
12     cout << "date2 = " << date2;
13     // 할당 연산자와 후위 증가 연산자 사용
14     Date date3 = date1;
15     Date date4 = date2;
16     date3++;
17     date4++;
18     cout << "date3 = " << date3;
19     cout << "date4 = " << date4;
20     // 복합 할당 연산자 사용
21     date3 += 20;
22     date4 -= 130;
23     cout << "date3 + 20 = " << date3;
24     cout << "date4 - 130 = " << date4;
25     // 날짜 차이 계산
26     cout <<"date3 - date4 = "
27                 << date3 - date4 << " days.";
```

```
28      return 0;
29  }
```

실행 결과

```
date1 = Sat Feb 8 2014
date2 = Sun Oct 15 1944
date3 = Sun Feb 9 2014
date4 = Mon Oct 16 1944
date3 + 20 = Sat Mar 1 2014
date4 - 130 = Thr Jun 8 1944
date3 - date4 = 25468 days.
```

3 연산자 오버로드를 활용해 Poly 클래스 만들기

컴퓨터 과학에서는 네트워크, 네트워크 보안 등에서 다항식을 굉장히 많이 활용합니다. 변수 하나를 갖는 다항식은 다음과 같이 정의할 수 있습니다. n은 지수이고, a는 계수입니다.

$$a_nx^n + a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0$$

표현

일단 다항식의 용어를 정리해봅시다. 어떤 다항식이 있을 때, 각각의 부분을 항term이라고 부릅니다. 예를 들어 $3.0x^5$ 등이 항입니다. $3.0x^5$은 계수coefficient가 3.0이고, 지수exponent가 5입니다. 가장 큰 지수를 차수degree라고 부릅니다. 구현을 할 때는 계수는 double 자료형, 지수는 unsigned int 자료형으로 만들면 될 것입니다. 이러한 항을 여러 개 갖는 다항식은 크게 3가지 방법으로 구현할 수 있습니다.

첫 번째 방법은 다항식의 차수만큼의 배열을 스택에 만들고, 배열의 인덱스를 지수, 해당 인덱스 내부에 들어있는 값을 계수로 활용하는 것입니다. 하지만 다항식의 최대 크기가 어느 정도 될지 모르므로, 아주 큰 배열을 만들어서 활용해야 합니다. 따라서 메모리 낭비가 발생하는 방법입니다.

두 번째 방법은 계수와 차수라는 데이터 멤버를 갖는 구조체를 만들고, 18장에서 설명하는 포인터를 활용한 링크드 리스트를 만들어 활용하는 방법입니다.

세 번째 방법은 첫 번째 방법보다는 효율적이지만, 두 번째 방법보다는 비효율적입니다. 다항식은 크기가 '차수 + 1'인 배열로 구현할 수 있습니다. 힙에 배열을 만들고, 포인터로 가리킨다

면 다항식별로 다른 크기의 배열을 갖게 만들 수 있습니다. 이러한 배열의 인덱스를 지수, 해당 인덱스 내부에 들어있는 값을 계수로 활용하는 것입니다. [그림 13-13]은 이를 그림으로 나타낸 것입니다. 다항식의 차수가 5라면, 배열의 크기가 6개 필요하다는 것에 주의하세요.

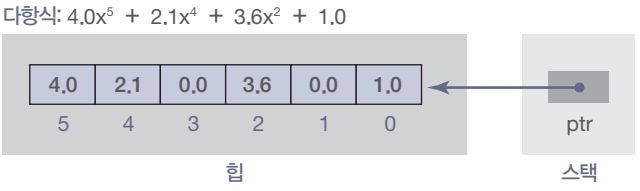

그림 13-13 차수가 5인 다항식의 표현

연산

다항식에 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지라는 5가지 연산을 정의합니다. 예를 들어 다음과 같은 다항식이 있다고 합시다.

```
poly1 = +4.00x⁵ + 2.00x³ + 5.00x² + 1.00x¹ + 4.00
poly2 = +2.00x² + 6.00
```

덧셈, 뺄셈, 곱셈, 나눗셈은 다음과 같이 이루어집니다.

```
poly1 + poly2 = +4.00x⁵ + 2.00x³ + 7.00x² + 1.00x¹ + 10.00
poly1 - poly2 = +4.00x⁵ + 2.00x³ + 3.00x² + 1.00x¹ - 2.00
poly1 * poly2 = +8.00x⁷ + 28.00x⁵ + 10.00x⁴ + 14.00x³ + 38.00x² + 6.00x¹ + 24.00
poly1 / poly2 = +2.00x³ - 5.00x¹ + 2.50
poly1 % poly2 = +31.00x¹ - 11.00
```

인터페이스 파일

지금까지 설명한 내용을 기반으로 다항식을 나타내는 Poly 클래스를 만들어봅시다. 인터페이스 파일, 구현 파일, 애플리케이션 파일을 분리해서 만듭니다.

[프로그램 13-16]은 Poly 클래스의 인터페이스 파일입니다. 3개의 생성자, 소멸자, 복사 생성자, 할당 연산자를 정의했습니다. 참고로 세 번째 생성자는 단항식 생성을 목적으로 만든 생성자입니다. 또한 +, −, *, /, %, 〈〈 연산자를 오버로드했습니다. 멤버 함수 fill은 다항식의 계수를 입력받을 때 사용합니다. 멤버 함수 max는 두 다항식의 최대 차수를 찾을 때 사용합니다.

프로그램 13-16 인터페이스 파일(poly.h) Prg13-16.cpp

```
1  /*************************************************************************
2   * Poly 클래스의 인터페이스 파일                                         *
3   *************************************************************************/
4  #ifndef POLY_H
5  #define POLY_H
6  #include <iostream>
7  #include <string>
8  #include <cassert>
9  #include <iomanip>
10 using namespace std;
11
12 // Poly 클래스
13 class Poly
14 {
15     private:
16         int degree;
17         double* ptr;
18     public:
19         Poly();
20         Poly(int degree);
21         Poly(int degree, double coef);   // 항이 하나인 다항식
22         ~Poly();
23         Poly(const Poly& origin);
24         Poly& operator=(const Poly& right);
25         void fill();
26         int max(int x, int y);
27         friend const Poly operator+(const Poly& left, const Poly& right);
28         friend const Poly operator-(const Poly& left, const Poly& right);
29         friend const Poly operator*(const Poly& left, const Poly& right);
30         friend const Poly operator/(const Poly& left, const Poly& right);
31         friend const Poly operator%(const Poly& left, const Poly& right);
32         friend ostream& operator<<(ostream& left, const Poly& poly);
33 };
34 #endif
```

구현 파일

[프로그램 13-17]은 Poly 클래스의 멤버 함수 선언을 구현하는 부분입니다.

프로그램 13-17 구현 파일(poly.cpp) Prg13-17.cpp

```
1   /*************************************************************************
2    * Poly 클래스의 구현 파일                                               *
3    *************************************************************************/
4   #include "poly.h"
5
6   // 기본 생성자
7   Poly::Poly()
8   :degree(0)
9   {
10      ptr = 0;
11  }
12  // 매개변수가 있는 생성자
13  Poly::Poly(int deg)
14  :degree(deg)
15  {
16      ptr = new double[degree + 1];
17      for(int i = degree; i >=0; i--)
18      {
19          ptr[i] = 0.0;
20      }
21  }
22  // 항이 하나인 다항식을 만드는 생성자
23  Poly::Poly(int deg, double coef)
24  :degree(deg)
25  {
26      ptr = new double[degree + 1];
27      for(int i = degree; i >=0; i--)
28      {
29          ptr[i] = 0.0;
30      }
31      ptr[degree] = coef;
32  }
33  // 소멸자
34  Poly::~Poly()
35  {
36      delete[] ptr;
37  }
38  // 복사 생성자
39  Poly::Poly(const Poly& origin)
40  {
41      ptr = new double[degree + 1];
```

```
42      for(int i = origin.degree; i >= 0; i--)
43      {
44          ptr[i] = origin.ptr[i];
45      }
46  }
47  // 할당 연산자
48  Poly& Poly::operator=(const Poly& right )
49  {
50      this->degree = right.degree;
51      this->ptr = new double[degree + 1];
52      for(int i = right.degree; i >= 0; i--)
53      {
54          (this->ptr)[i] = right.ptr[i];
55      }
56      return *this;
57  }
58  // + 연산자
59  const Poly operator+(const Poly& left, const Poly& right)
60  {
61      Poly result(max(left.degree, right.degree));
62      for(int i = result.degree; i >= 0; i--)
63      {
64          if(i <= left.degree && i <= right.degree)
65          {
66              result.ptr[i] = left.ptr[i] + right.ptr[i];
67          }
68          else if(i <= left.degree && i > right.degree)
69          {
70              result.ptr[i] = left.ptr[i];
71          }
72          else
73          {
74              result.ptr[i] = right.ptr[i];
75          }
76      }
77      return result;
78  }
79  // - 연산자
80  const Poly operator-(const Poly& left, const Poly& right)
81  {
82      Poly result(max(left.degree , right.degree));
83      for(int i = result.degree; i >= 0; i--)
```

```
84      {
85          if(i <= left.degree && i <= right.degree)
86          {
87              result.ptr[i] = left.ptr[i] - right.ptr[i];
88          }
89          else if(i <= left.degree && i > right.degree)
90          {
91              result.ptr[i] = left.ptr[i];
92          }
93          else
94          {
95              result.ptr[i] = -right.ptr[i];
96          }
97      }
98      return result;
99  }
100 // * 연산자
101 const Poly operator*(const Poly& left, const Poly& right)
102 {
103     int degree = left.degree + right.degree;
104     Poly result(degree);
105     for(int i = result.degree; i >= 0; i--)
106     {
107         result.ptr[i] = 0;
108     }
109     for(int i = left.degree; i >= 0; i--)
110     {
111         for(int j = right.degree; j >= 0; j--)
112         {
113             result.ptr[i + j] +=(left.ptr[i] * right.ptr[j]);
114         }
115     }
116     return result;
117 }
118 // / 연산자
119 const Poly operator/(const Poly& left, const Poly& right)
120 {
121     Poly result(left.degree - right.degree);
122     Poly temp(left.degree);
123     temp = left;
124     int i = temp.degree;
125     int j = right.degree;
```

```
126     int k = i - j;
127     while(i >= j)
128     {
129         double coef = temp.ptr[i] / right.ptr[j];
130         Poly poly(k , coef);
131         temp = temp - (poly * right);
132         result = result + poly;
133         i--;
134         k = i - j;
135     }
136     return result;
137 }
138 // % 연산자
139 const Poly operator%(const Poly& left, const Poly& right)
140 {
141     Poly result(left.degree - right.degree - 1);
142     Poly temp(left.degree);
143     temp = left;
144     result = temp - (temp /right) * right;
145     return result;
146 }
147 // << 연산자
148 ostream& operator<<(ostream& output, const Poly& poly)
149 {
150     string sign;
151     for(int i = poly.degree; i >= 0; i--)
152     {
153         if(poly.ptr[i] > 0.0 || poly.ptr[i] < 0.0)
154         {
155             output << fixed << showpos << setprecision(2);
156             output << poly.ptr[i];
157             output << noshowpos;
158             if(i != 0)
159             {
160                 output << "x^";
161                 output << i;
162             }
163             output << " ";
164         }
165     }
166     return output;
167 }
```

```
168 // 헬퍼 함수
169 int max(int x, int y)
170 {
171     if(x >= y)
172     {
173         return x;
174     }
175     return y;
176 }
177 // 다항식을 입력받는 함수
178 void Poly::fill()
179 {
180     for(int i = degree; i >= 0; i--)
181     {
182         cout << i << "차수의 계수를 입력하세요: ";
183         cin >> ptr[i];
184     }
185     cout << endl;
186 }
```

애플리케이션 파일

[프로그램 13-18]은 다항식 연산을 위해 만든 Poly 클래스의 동작을 테스트하는 애플리케이션 파일입니다.

프로그램 13-18 애플리케이션 파일(app.cpp) Prg13-18.cpp

```
1  /*************************************************************
2   * Poly 클래스를 사용하는 애플리케이션 파일                  *
3   *************************************************************/
4  #include "poly.h"
5
6  int main()
7  {
8      // 다항식 생성하고 입력받기
9      Poly poly1(5);
10     poly1.fill();
11     Poly poly2(2);
12     poly2.fill();
13     // 출력
14     cout << "다항식의 상태 " << endl;
```

```
15      cout << "Poly1 = " << poly1 << endl;
16      cout << "Poly2 = " << poly2 << endl << endl;
17      // 연산자 사용
18      Poly poly3 = poly1 + poly2;
19      Poly poly4 = poly1 - poly2;
20      Poly poly5 = poly1 * poly2;
21      Poly poly6 = poly1 / poly2;
22      Poly poly7 = poly1 % poly2;
23      // 결과 출력
24      cout << "다항식 연산 결과" << endl;
25      cout << "Poly1 + Poly2 = " << poly3 << endl;
26      cout << "Poly1 - Poly2 = " << poly4 << endl;
27      cout << "Poly1 * Poly2 = " << poly5 << endl;
28      cout << "Poly1 / Poly2 = " << poly6 << endl;
29      cout << "Poly1 % Poly2 = " << poly7 << endl;
30      return 0;
31  }
```

실행 결과

```
4차수의 계수을 입력하세요: 4
5차수의 계수을 입력하세요: 0
3차수의 계수을 입력하세요: 2
2차수의 계수을 입력하세요: 5
1차수의 계수을 입력하세요: 1
0차수의 계수을 입력하세요: 4

2차수의 계수을 입력하세요: 2
1차수의 계수을 입력하세요: 0
0차수의 계수을 입력하세요: 6

다항식의 상태
Poly1 = +4.00x^5 +2.00x^3 +5.00x^2 +1.00x^1 +4.00
Poly2 = +2.00x^2 +6.00

다항식 연산 결과
Poly1 + Poly2 = +4.00x^5 +2.00x^3 +7.00x^2 +1.00x^1 +10.00
Poly1 - Poly2 = +4.00x^5 +2.00x^3 +3.00x^2 +1.00x^1 -2.00
Poly1 * Poly2 = +8.00x^7 +28.00x^5 +10.00x^4 +14.00x^3 +38.00x^2 +6.00x^1 +24.00
Poly1 / Poly2 = +2.00x^3 -5.00x^1 +2.50
Poly1 % Poly2 = +31.00x^1 -11.00
```

참고로 몫(Poly1 / Poly2)에 제수(Poly2)를 곱하고, 결과를 나머지(Poly1 % Poly2)와 더하면 피제수(Poly1)가 됩니다.

CHAPTER 16

입출력 스트림

학습 목표

- 스트림 객체를 사용해서 데이터 소스 또는 데이터 싱크와 프로그램이 데이터를 전달하는 방법을 알아봅니다.
- 스트림 클래스의 계층에서 최상위에 있는 인스턴스화할 수 없는 가상 클래스인 ios 클래스를 알아봅니다.
- C++이 미리 인스턴스화해서 제공하는 콘솔스트림에 대해서 알아봅니다.
- 파일을 데이터 소스 또는 싱크로 활용할 때 사용할 수 있는 파일 스트림에 대해서 알아봅니다.
- 애플리케이션과 문자열 클래스 사이의 어댑터로 사용할 수 있는 문자열 스트림에 대해서 알아봅니다.
- ios 클래스에 정의된 플래그와 필드를 사용해서 데이터를 형식화하는 방법과, 표준 조정자와 사용자 정의 조정자를 사용해서 데이터를 형식화하는 방법을 이해합니다.

06 [Lab] 응용 프로젝트

1 파일 정렬해서 결합하기

작은 것에서 큰 순서로 정수가 정렬된 상태로 들어있는 두 파일이 있습니다. 파일을 결합하여 정렬을 다시 하면 어떨까요? 그림으로 정리해보면 [그림 16-15]와 같습니다. infile은 입력 파일, outfile은 출력 파일입니다.

그림 16-15 파일을 정렬해서 결합하는 흐름

[그림 16-15]에서 두 파일을 결합하는 과정은 반복문 13회를 돌면서 이루어집니다. 각각의 반복에서 변수 first 또는 second의 값이 outfile로 복사됩니다. [프로그램 16-29]는 이를 구현한 코드입니다. 14행에서 변수 first와 second를 선언합니다. 두 변수는 처리 과정 중에 첫 번째 파일과 두 번째 파일의 숫자를 잠시 갖고 있을 변수입니다. 15행에서는 센티널로 활용할 변수 sentinel을 선언했습니다. 이는 입력되는 숫자가 다룰 수 없는 범위라면 이를 보정하기 위해서 사용합니다.

프로그램 16-29 파일 정렬해서 결합하기 Prg16-29.cpp

```
/*********************************************************************
 * 두 파일 내부의 정수를 결합하고                                    *
 * 정렬해서 출력하는 프로그램                                        *
 *********************************************************************/
#include <iostream>
#include <fstream>
#include <assert.h>
#include <limits>
using namespace std;

int main()
{
    // 선언과 초기화
    int first, second;
    int sentinel = numeric_limits<int>::max();
    // 스트림 인스턴스화하고 파일 열기
    ifstream strm1("infile1");
    ifstream strm2("infile2");
    ofstream strm3("outfile");
    if(!strm1.is_open())
    {
        cout << "infile1을 열 때 오류가 발생했습니다." << endl;
        assert(false);
    }
    if(!strm2.is_open())
    {
        cout << "infile2을 열 때 오류가 발생했습니다." << endl;
        assert(false);
    }
    if(!strm3.is_open())
    {
        cout << "outfile을 열 때 오류가 발생했습니다." << endl;
        assert(false);
    }
    // 처리
    strm1 >> first;
    strm2 >> second;
    while(strm1 || strm2)
    {
        if(first <= second)
        {
```

```
42              strm3 << first << " ";
43              strm1 >> first;
44              if(!strm1)
45              {
46                  first = sentinel;
47              }
48          }
49          else
50          {
51              strm3 << second << " ";
52              strm2 >> second;
53              if(!strm2)
54              {
55                  second = sentinel;
56              }
57          }
58      }
59      // 파일 닫기
60      strm1.close();
61      strm2.close();
62      strm3.close();
63      return 0;
64  }
```

2 대칭 키 암호화 구현하기

파일을 통해서 정보를 전달할 때는 항상 보안을 고려해야 합니다. 이러한 목적으로 사용되는 가장 일반적인 기술은 바로 암호화cryptography입니다. 암호화는 비밀 키라는 것을 사용해 일반 텍스트plain text를 비밀 텍스트cipher text로 변환해주는 기술입니다.

역자 암호학에서는 plain text를 평문이라고도 표현합니다.

암호화는 기본적으로 대칭 키 암호화와 비대칭 키 암호화라는 2가지 방법이 있습니다. 이번 절에서는 대칭 키 암호화를 살펴봅니다. 대칭 키 암호화에서는 암호를 걸 때와 해제할 때 모두 같은 키를 사용합니다. 그래서 대칭이라는 이름이 붙은 것입니다. [그림 16-16]은 대칭 키 암호화의 기본적인 개념을 그림으로 나타낸 것입니다.

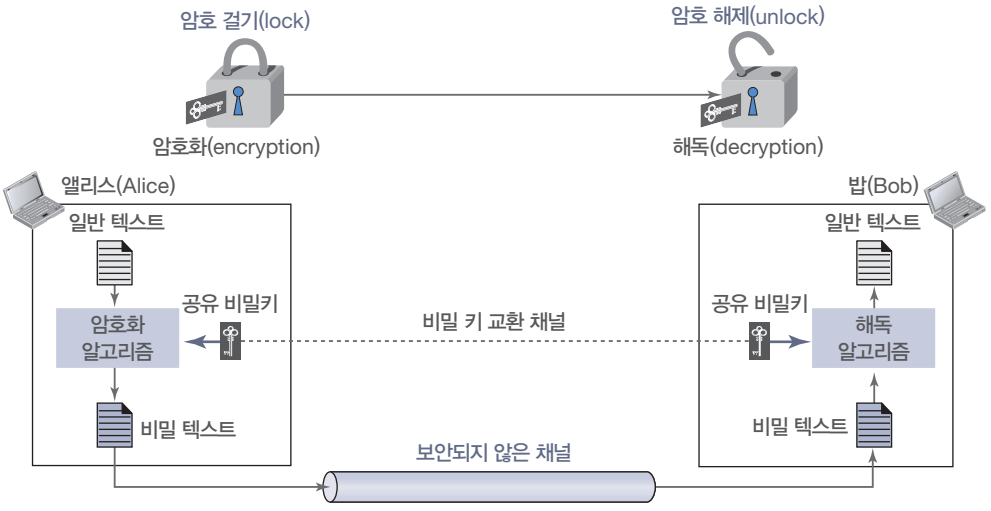

그림 16-16 대칭 키 암호화의 기본적인 개념

[그림 16-16]은 Alice가 보안되지 않은 채널을 통해서 Bob에게 메시지를 전달하는 과정입니다. 이렇게 메시지를 전달하면 보안되지 않은 채널로 암호화가 완료된 비밀 텍스트가 전달되기 때문에 보안 위협을 줄일 수 있습니다. Alice는 암호화 알고리즘encryption algorithm과 공유 비밀키shared secret key를 사용합니다. Bob은 비밀 텍스트를 일반 텍스트로 변환해서 Alice의 메시지를 읽어야 합니다. 이때 Bob은 해독 알고리즘decryption algorithm에 Alice와 공유하고 있는 공유 비밀 키를 사용합니다.

대칭 키 암호화는 전통적인 방법과 현대적인 방법으로 구분할 수 있습니다. 전통적인 대칭 키 암호화는 굉장히 간단하게 텍스트를 기반으로 암호화를 하는 방법입니다(현대에는 사용하지 않음). 현대적인 대칭 키 암호화는 비트 기반으로 암호화를 해서 조금은 더 안전합니다. 우리는 이번 절에서 전통적인 방법을 구현합니다. DES Data Encryption Standard와 같은 현대적인 방법은 높은 수준의 암호화 이론과 수 이론을 알고 있어야 하므로 이 책에서는 생략하겠습니다. 관심이 있다면 암호학을 더 공부해보기 바랍니다.

전통적인 대칭 키 알고리즘 중에서 모노 알파벳 암호화monoalphabetic cipher라는 것이 있습니다. 이는 일반 텍스트의 문자를 위치와 상관 없이 항상 동일한 문자로 변환합니다. 예를 들어서 A라는 글자를 D로 변환한다면, 파일 전체에서 예외 없이 A라는 글자를 D로 변환합니다. 일반 텍스트의 문자와 비밀 텍스트의 문자가 일대일로 대응되므로 모노mono라는 이름이 붙은 것입니다. 이러한 변환을 위해서 일반적으로 2차원 표를 사용합니다. 추가적으로 일반 텍스트의 소문자는 비밀 텍스트에서 대문자로 표현되게 만듭니다. [그림 16-17]은 변환 표의 예를 나타낸 것입니다. [프로그램 16-30]에서 클래스를 구현합니다.

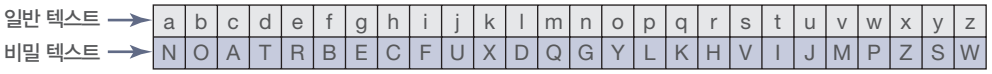

그림 16-17 모노 알파벳 변환에 사용되는 변환 표의 예

프로그램 16-30 인터페이스 파일(monoalpha.h) Prg16-30.cpp

```
1  /*************************************************************************
2   * MonoAlpha 클래스의 인터페이스 파일                                    *
3   *************************************************************************/
4  #ifndef MONOALPHA_H
5  #define MONOALPHA_H
6  #include <iostream>
7  using namespace std;
8
9  class MonoAlpha
10 {
11     private:
12         static const char key[][2];
13         char searchEncrypt(char c);
14         char searchDecrypt(char c);
15     public:
16         MonoAlpha();
17         ~MonoAlpha();
18         void encrypt(const char* plainFile, const char* cipherFile);
19         void decrypt(const char* cipherFile, const char* plainFile);
20 };
21 #endif
```

정적 2차원 배열과 비밀키를 포함하는 MonoAlpha 클래스를 만들었습니다. 이 클래스에는 encrypt와 decrypt라는 public 멤버 함수가 있습니다. 그리고 이러한 public 멤버 함수에서 알파벳 변환을 할 때 활용하는 searchEncrypt와 searchDecrypt라는 private 멤버 함수가 있습니다.

이제 구현 파일을 살펴봅시다. encrypt와 decrypt 함수에서는 입력 파일에서 문자를 하나씩 추출하고, searchEncrypt와 searchDecrypt 함수를 호출해서 문자를 변환하여 출력 파일에 씁니다.

프로그램 16-31 구현 파일(monoalpha.cpp) Prg16-31.cpp

```
1  /*************************************************************************
2   * MonoAlpha 클래스의 구현 파일                                          *
```

```
3   ********************************************************************/
4   #include "monoalpha.h"
5   #include<fstream>
6
7   // 생성자
8   MonoAlpha::MonoAlpha()
9   {
10  }
11  // 소멸자
12  MonoAlpha::~MonoAlpha()
13  {
14  }
15  // public 멤버 함수
16  void MonoAlpha::encrypt(const char* plainFile, const char* cipherFile)
17  {
18      ifstream istrm(plainFile, ios::in);
19      ofstream ostrm(cipherFile, ios::out);
20      char c1, c2;
21      while(istrm.get(c1))
22      {
23          c2 = searchEncrypt(c1);
24          ostrm.put(c2);
25      }
26      istrm.close();
27      ostrm.close();
28  }
29  // public 멤버 함수
30  void MonoAlpha::decrypt(const char* cipherFile, const char* plainFile)
31  {
32      ifstream istrm(cipherFile, ios::in);
33      ofstream ostrm(plainFile, ios::out);
34      char c1, c2;
35      while(istrm.get(c1))
36      {
37          c2 = searchDecrypt(c1);
38          ostrm.put(c2);
39      }
40      istrm.close();
41      ostrm.close();
42  }
43  // private 멤버 함수
44  char MonoAlpha::searchEncrypt(char c)
```

```
{
    int i = 0;
    while(true)
    {
        if(key[i][0] == c)
        {
            return key[i][1];
        }
        i++;
    }
}
// private 멤버 함수
char MonoAlpha::searchDecrypt(char c)
{
    int i = 0;
    while(true)
    {
        if(key[i][1] == c)
        {
            return key[i][0];
        }
        i++;
    }
}
// 정적 키 배열 정의
const char MonoAlpha::key[][2] = {{'a', 'N'}, {'b', 'N'},
   {'c', 'A'}, {'d', 'T'}, {'e', 'R'}, {'f', 'B'}, {'g', 'E'}, {'h', 'C'},
   {'i', 'F'}, {'j', 'U'}, {'k', 'X'}, {'l', 'D'}, {'m', 'Q'}, {'n', 'G'},
   {'o', 'Y'}, {'p', 'L'}, {'q', 'K'}, {'r', 'H'}, {'s', 'V'}, {'t', 'I'},
   {'u', 'J'}, {'v', 'M'}, {'w', 'P'}, {'x', 'Z'}, {'y', 'S'}, {'z', 'W'}};
```

암호화하는 애플리케이션 파일과 암호를 해제하는 애플리케이션 파일이 필요합니다. [프로그램 16-32]는 암호화하는 애플리케이션입니다. Alice는 문자열을 암호화해서 Bob에게 보냅니다. 변환을 쉽게 하고자, 그리고 코드의 보안을 강화하기 위해서 문자열에는 소문자만 사용하게 했고, 암호화한 문자열에는 대문자만 사용하게 했습니다. 또한 마침표를 사용하는 것도 해커에게 일말의 단서가 될 수 있는 부분이므로 사용하지 않게 했습니다.

프로그램 16-32 암호화하는 app1.cpp 파일 Prg16-32.cpp

```
/*************************************************************************
 * 메시지 암호화에 사용하는 애플리케이션 파일                              *
```

```
 ********************************************************************/
#include "monoalpha.h"

int main()
{
    MonoAlpha monoalpha;
    monoalpha.encrypt("plainFile", "cipherFile");
    return 0;
}
```

실행 결과

```
일반 텍스트 파일의 내용:
thisisthefiletoencrypt
비밀 텍스트 파일의 내용:
ICFVFVICRBFDRIYRGAHSLI
```

[프로그램 16-33]은 암호화를 해제하는 애플리케이션입니다. Bob은 Alice에게서 받은 암호화된 문자를 이 프로그램으로 다시 원래 문자열로 변환할 수 있습니다.

프로그램 16-33 암호화를 해제하는 app2.cpp 파일 Prg16-33.cpp

```
/********************************************************************
 * 메시지 해독에 사용하는 애플리케이션 파일                          *
 ********************************************************************/
#include "monoalpha.h"

int main()
{
    MonoAlpha monoalpha;
    monoalpha.decrypt("cipherFile", "plainFile");
    return 0;
}
```

실행 결과

```
비밀 텍스트 파일의 내용:
ICFVFVICRBFDRIYRGAHSLI
일반 텍스트 파일의 내용:
thisisthefiletoencrypt
```

참고로 monoalphabetic 프로그램은 교육 목적의 프로그램입니다. 따라서 실제 시스템에서 이를 활용해서 암호화하는 것은 보안 상 위험합니다.