연습문제 이론 정답

1장 연습문제

이론 문제

1. ② 소프트웨어 기업

2. ④ 기계어를 익혀서 프로그램을 작성하는 것이 좋은 개발자가 되는 지름길이다.

3. ④ Fortran

4. ② C

5. ③ 운영체제 독립적인 프로그램 작성이 필요해서

6. ② 흐름도

7. ① C 언어로 작성된 프로그램과 호환성을 가질 수 있다.

8. ④ 전역 변수가 존재하게 되어 캡슐화의 원칙이 무너졌다.

9. C++은 C 언어와의 호환성을 목표로 하여 설계되었으며 객체 지향 개념을 추가하였다. 그러나 클래스 바깥에 전역 변수나 함수들을 선언할 수 있게 함으로써 객체 지향의 핵심 개념인 캡슐화의 원칙이 무너지게 되었다.

10. ② 호환성

11. ③ 인라인 함수

12. (1) 캡슐화 : TV 객체는 플라스틱 외관으로 둘러싸고 있어서 내부의 구조나 구성을 알 수 없고 직접 건드릴 수 없다. 다만 외부와의 접촉(통신)을 위해 On/OFF 버튼, 볼륨 제어 버튼, 신호 입출력 아웃렛 등이 외부에 노출되어 있다.
    (2) 다형성 : > 연산자는 5 > 2와 같이 수의 비교에도 사용되며, Circle 타입의 객체 pizza와 donut에 대해 누가 큰 객체인지 비교하기 위해 pizza > donut의 연산으로 사용할 수도 있다.
    (3) 상속성 : 메뚜기는 태어나고 자라고 죽는 생명 주기를 가진 생물의 속성을 물려받

았다.

13. 다형성(구체적으로 함수 중복)

14. ④ 타입 변환

15. ③ C++ 프로그램은 C 소스 프로그램을 수용하여 사용할 수 있지만, 이미 컴파일된 C 언어의 목적 파일은 링크시켜 사용할 수 없다.

16. ① C++ 소스 파일은 텍스트 파일이 아니라 바이너리 파일이다.

17. ③ 링킹은 C++ 프로그램 개발 과정에서 실행 파일을 만들기 위해 반드시 필요하다.

18. hello.obj 파일은 C++ 라이브러리에 있는 << 연산자 코드와 cout 객체 코드를 필요로 한다. 링킹은 C++ 표준 라이브러리로부터 이들 코드만 뽑아내어 hello.obj와 합쳐 hello.exe를 만들어 내는 과정이다.

19. ④ 실행 파일의 확장자는 운영체제에 관계없이 .exe로 표준화되어 있다.

20. ① C++ 통신 라이브러리

21. 최근에는 동일한 프로그램 코드에 구체적인 타입을 적용할 수 있도록 함수나 클래스를 일반화시킨 제네릭(혹은 템플릿) 함수와 제네릭(혹은 템플릿) 클래스를 이용하여 프로그램을 작성하는 새로운 프로그래밍 패러다임인 제네릭 프로그래밍(혹은 일반화 프로그래밍)이 활발히 사용되고 있다. 이것은 흐름도를 중심으로 작업의 실행 순서에 따라 프로그램을 작성하는 절차 지향 프로그래밍 기법에서, 객체 사이의 상호 작용과 관계를 중심으로 프로그램을 작성하는 객체 지향 프로그래밍 기법 이후에 도입된 것이다.

22. C++ 프로그램의 편집, 컴파일, 링킹, 디버깅 등 C++ 프로그램 개발에 필요한 모든 소프트웨어 도구를 제공하는 통합 소프트웨어를 말한다.

2장 연습문제

이론 문제

1. `int main()`

2. ③

3. 5번 라인. `return 0;`

4. (1) C 언어에서는 모든 변수 선언이 반드시 실행문 전에 이루어져야 하므로, `int square = a * a;` 문장에서 컴파일 오류가 발생한다. 변수 square가 실행문 앞에 선언되어야 한다.
   (2) C++에서는 실행문 중간에도 변수의 선언이 가능하다.
   (3) 변수가 필요한 위치에 선언하면, 이미 선언된 비슷한 이름의 다른 변수로 이름을 잘못 타이핑하는 오류를 발견하기 쉬운 반면, 변수들이 함수 전체에 흩어져 있어 변수를 찾기가 용이하지 않는 단점도 있다.

5. 화면에 다음과 같이 출력된다.

```
I love C++
I love programming
```

6. (1) `using std::cout;`
   (2) `using namespace std;`

7. (1) `#include <iostream>; -> #include <iostream>`
   (2) `using namespace std -> using namespace std;`
   (3) `std::cin << name; -> std::cin >> name;`
   (4) `std:cout << 1 << 2 << 'a' << "hello" << '\n';`
   `-> std::cout << 1 << 2 << 'a' << "hello" << '\n';`

8. (1) `using std::cin;` (틀린 부분 없다.)
   (2) `int year = 1; // year은 년도를 나타낸다.` (틀린 부분 없다.)
   (3) `int n=1; cout >> n+200; -> int n=1; cout << n+200;`
   (4) `int year = 2014; cout << 2014+"년"; -> int year = 2014; cout << 2014 << "년";`

9. #include "myheader.h"

10. (1) (○)
(2) (○)
(3) (○)
(4) (×)
(5) (×)

11. <cstring>

12. (1) Kitae님 환영합니다
(2) Kitae님 환영합니다

13. ④

14. ①. cin >> buf; 로는 빈칸이 들어 있는 문자열을 읽을 수 없기 때문

15. namespace

16. std

17. std

18. <iostream> 헤더 파일

19.

```
#include <iostream>
using namespace std;
int main() {
   int age = 20;
   char* pDept = "컴퓨터 공학과";
   cout << age << " " << pDept;
}
```

20.

```
#include <iostream>
using namespace std;
int main() {
   for(int n=0; n<4; n++) {
      for(int m=0; m<n+1; m++)
         cout << '*';
      cout << endl;
   }
}
```

3장 연습문제

이론 문제

1. 객체의 캡슐화는 객체 외부의 접근으로부터 객체를 보호하기 위한 것이다.

2. ③. 클래스의 멤버들은 private으로 선언하는 것이 바람직하다.

3. 이 코드는 캡슐화를 이루고 있지 못하다. acc 변수와 add() 함수는 어떤 클래스에도 포함되어 있지 않아 누구나 이들을 접근할 수 있기 때문이다. 또한 Circle 클래스의 멤버 변수 radius가 public 속성으로 되어 있는데 적절치 않다. 캡슐화를 위해 수정하면 다음과 같다.

```
class Count {
   int acc;
public:
   Count(int a) { acc = a; }
   int add(int x) {
      acc += x;
      return acc;
   }
   int getAcc() { return acc; }
};

class Circle {
   int radius;
public:
   double getArea();
};
```

4.

```
class Age {
   int age;
public:
   Age(int a) { age = a; }
   void older() {
      age++;
   }
```

```
   int getAge() { return age; }
};
class Circle {
   int radius;
public:
   double getArea();
};
```

5. 클래스의 선언부는 ;으로 마쳐야 한다.

```
class Circle {
   int radius;
   double getArea();
}; // 세미콜론(;) 추가
```

6. 클래스의 멤버 변수는 선언 시에 초기화할 수 없다.

```
int height = 20;  -> int height;
```

7. main()의 다음 줄은 Building 클래스의 기본 생성자가 없기 때문에 컴파일 오류가 발생한다.

```
Building twin, star;
```

컴파일 오류를 해결하기 위해 Building 클래스에 다음과 같이 기본 생성자를 추가한다.

```
class Building {
private:
   int floor;
public:
   Building() { floor = 0; } // 기본 생성자 추가
   Building(int s) { floor = s; }
};
```

8.

```
class Calendar {
private:
   int year;
```

```
public:
   Calendar();
   int getYear();
};
Calendar::Calendar() {
   year = 10;
}
int Calendar::getYear() {
   return year;
}
```

9. ②. 생성자는 매개 변수 타입이나 개수가 다르면 여러 개 중복 작성 가능하다.

10. ③. 소멸자는 매개 변수를 가지지 않는다.

11. (1)과 (2)를 합쳐서 구현하면 다음과 같다.

```
#include <iostream>
using namespace std;
class House {
   int numOfRooms;
   int size;
public:
   House(int n, int s);
   ~House(); // (2)의 답
};

House::House(int n, int s) { // (1)의 답
   numOfRooms = n; size = s;
   cout << "생성 " << n << ' ' << s << endl;
}

House::~House() { // (2)의 답
   cout  << "소멸 " << numOfRooms << ' ' << size << endl;
}

void f() {
   House a(2,20);
}
House b(3,30), c(4,40);
int main() {
   f();
   House d(5,50);
}
```

(3) b생성 > c생성 > a생성 > a소멸 > d생성 > d소멸 > c소멸 > b소멸

12. c생성 > b생성 > a생성 > a소멸 > b소멸 > c소멸

13. TV() 생성자가 private으로 선언되어 있기 때문에, main() 함수의 TV LG; 라인에서 컴파일 오류가 발생한다. 다음과 같이 TV 클래스를 수정하면 된다.

```
class TV {
public:
   int channels;
   TV() { channels = 256; }
   TV(int a) { channels = a; }
};
```

14. main() 함수에서 LG.channels = 200;에서 컴파일 오류가 발생한다. channels가 private으로 선언되어 있기 때문에 main()에서는 접근할 수 없다. 다음과 같이 코드를 수정하는 것이 바람직하다.

```
class TV {
   int channels;
   int colors;
public:
   TV() { channels = 256; }
   void setChannels(int n) { channels = n; }
   void setColors(int n) { colors = n; }
   TV(int a, int b) { channels = a; colors = b; }
};

int main() {
   TV LG;
   LG.setChannels(200);
   LG.setColors(60000);
   TV Samsung(100, 50000);
}
```

15. 클래스 선언부에 구현된 멤버 함수는 자동 인라인으로 처리된다.

```
TV() { channels = 256; }
TV(int a) { channels = a; }
```

16. ②

17. ①

18. ①. ②는 재귀함수, ③은 반복문, ④는 static 변수를 가지고 있어서 컴파일러에 따라서는 인라인으로 처리하지 않을 가능성이 있다.

19. ④

20.

```
class Family {
   char tel[11];
public:
   int count;
   char address[20];
   Family();
};
```

21.

```
struct Universe {
   Universe();
private:
   char creator[10];
   int size;
   char dateCreated[10];
};
```

4장 연습문제

이론 문제

1. (1) Rect *p;
   (2) p = &r;
   (3) cout << p->getWidth() << 'x' << p->getHeight();

2. (1) q = new Rect(w, h);
   (2) cout << q->getArea() << endl;
   (3) delete q;

3. ①. 기본 생성자 Rect()이 없기 때문

4. 기본 생성자를 작성해야 한다. 예를 들면 다음과 같다.

```
Rect::Rect() { width = 1; height = 1; }
```

5.

```
Rect r[5] = { Rect(), Rect(2, 3), Rect(3,4), Rect(4,5), Rect(5,6) };
int sum = 0;
for(int i=0; i<5; i++)
   sum += r[i].getArea();
cout << sum;
```

6. ④

7. ④

8. 프로그램 실행 결과 다음과 같이 출력된다.

```
기본생성자
기본생성자
기본생성자
소멸자
소멸자
소멸자
```

9. ①

10. delete p; 문을 delete [] p;로 수정하여야 한다.

11. ③. this는 static 멤버 함수에서 사용할 수 없다.

12. ③. this는 생성자에서 사용할 수 있다.

13.

```
class Location {
   int width, height;
public:
   Location() { this->width = this->height = 0; }
   Location(int width, int height) {
      this->width = width; this->height = height;
   }
   void show();
};
void Location::show() {
   cout << this->width << this->height << endl;
}
```

14. 메모리 누수란 할당받은 동적 메모리에 대한 포인터를 잃어버리게 되어, 동적 메모리를 사용하지도 않고 반환할 수도 없게 된 상황에서 발생한다.

15. (1) 함수 f()가 종료하면 할당받은 new char[10]개의 메모리의 누수가 발생한다. 메모리 누수가 발생하지 않도록 수정하면 다음과 같다.

```
void f() {
   char *p = new char [10];
   strcpy(p, "abc");
   delete [] p;
}
```

(2) 메모리 누수가 발생하지 않는다.
(3) 메모리 누수가 발생하지 않는다.
(4) 메모리 누수가 발생한다. for문의 p = new int;가 반복되면 이전에 할당받은 int 메모리를 접근할 수 없게 되어 메모리 누수가 발생한다. 메모리 누수를 막기 위해

다음과 같이 코드를 수정하면 된다.

```
void f() {
   int *p;
   for(int i=0; i<5; i++) {
      p = new int; // 할당받고
      cin >> *p;
      if(*p % 2 == 1) {
         delete p; // 반환하고
         break;
      }
      delete p; // 반환하고
   }
}
```

16. ① <string>

17.

```
int n = stoi(s1);
int m = stoi(s2);
cout << n + m;
```

18. ③

19.

```
// string a("My name is Jane.");
string a = "My name is Jane.";

// char ch = a.at(2);
char ch = a[2];

// if(a.compare("My name is John.") == 0) cout << "same";
if(a == "My name is John.") cout << "same";

// a.append("~~");
a += "~~";

// a.replace(1, 1, "Y");
a[1] = 'Y';
```

5장 연습문제

이론 문제

1. ④

2. ①

3. 주소에 의한 호출

4. (1) 두 함수의 선언은 같다.
   (2) 두 함수의 선언은 다르다.

5. (1) 5가 출력된다. '값에 의한 호출'이므로 square()를 호출한 이후 m 값에는 변화 없음
   (2) 25가 출력된다. '참조에 의한 호출'이므로 square()에서 m 값을 변경함

6. 다음과 같이 배열 m의 제곱이 각각 출력된다.
   ```
   1 4 9
   ```

7. ②

8. ②

9. ①. 참조 변수는 다른 변수로 초기화되어야 한다.

10. (1) array [] = { 0, 2, 4, 6, 8, 10, 12, 14, 16, 100 } 값을 가진다.
    (2) array [] = { 0, 4, 6, 8, 10, 12, 14, 16, 18, 18 } 값을 가진다.
    (3) array [] = { 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 } 값을 가진다.
    (4) array [] = { 0, 2, 4, 6, 0, 10, 12, 14, 16, 18 } 값을 가진다. f(2)는 배열 array[2]의 참조를 리턴하므로 f(4) = 0;의 결과가 된다.

11. copy() 함수의 호출은 '값에 의한 호출'로, 매개 변수가 실인자 a, b에 영향을 주지 못한다. 그러므로 복사가 되지 않는다. copy()의 호출문을 그대로 두고, copy()를 '참조에 의한 호출'이 일어나도록 다음과 같이 작성하면 된다.

```
void copy(int& dest, int src) {
   dest = src;
}
```

다음과 같이 copy() 함수를 수정하는 '주소에 의한 호출'은 정답이 아니다.

```
void copy(int* dest, int* src) {
   *dest = *src;
}
```

왜냐하면 호출문도 다음과 같이 고쳐야 하기 때문이다.

```
int a=4, b=5;
copy(&a, &b); // b 값을 a에 복사
```

12. big1() 함수에서 매개 변수 a, b는 x, y와 다른 별도의 공간을 할당받는다. 그리고 큰 값 b의 참조를 리턴한다고 해도 b는 big1() 함수와 함께 소멸되어 버리기 때문에, z가 참조하는 변수는 없게 된다. big1() 호출 후 x=1, y=2로 그대로 남아 있고 z=100이 된다. 그러나 big2()의 경우 a, b 모두 참조 매개 변수이기 때문에, a는 곧 x이며, b는 곧 y이다. 그러므로 큰 값 b의 참조를 리턴하면 w는 곧 y에 대한 참조가 되고 w=100;은 y에 100을 기록하게 된다. big2() 호출 후 x=1, y=100, w=100이 된다.

13. 기본 생성자 MyClass(); 복사 생성자 MyClass(MyClass&);

14. ②

15. (1) 소멸자 함수는 다음과 같이 간단히 작성할 수 있다.

```
MyClass::~MyClass() { delete element; }
```

(2) 디폴트 복사 생성자는 모든 멤버를 1:1로 복사하도록 작성된다.

```
MyClass::MyClass(MyClass& b) {
   size = b.size;
   element = b.element;
}
```

(3) 깊은 복사 생성자는 멤버뿐 아니라 포인터가 멤버가 할당받은 동적 메모리를 따로 할당받고 동적 메모리의 내용까지도 복사한다.

```
MyClass::MyClass(MyClass& b) {
   size = b.size;
   element = new int [size]; // 배열을 따로 할당받는다.
   for(int i=0; i<size; i++) element[i] = b.element[i]; // 배열을 복사한다.
}
```

16. ①

17.

```
Student::Student(Student& b) {
   name = b.name;
   id = b.id;
   grade = b.grade;
}
```

18.

```
Student::Student(Student& b) {
   pName = b.pName;
   pId = b.pId;
   grade = b.grade;
}
```

19. 객체의 치환문 a = b;은 b의 각 멤버를 a의 각 멤버에 1:1로 복사하기 때문에 a.element 포인터는 b.element에 할당된 메모리를 가리키게 되어 a, b 객체는 동적 메모리를 공유하게 된다. 그러므로 객체 a의 element 배열에 대한 변화는 객체 b의 element 배열에 대한 변화로 나타나며, 그 반대도 마찬가지이다. 또한 main()이 종료될 때도 문제가 발생한다. 객체 b가 소멸될 때 소멸자에 의해 element 배열을 정상적으로 반환하지만, 다시 객체 a가 소멸될 때 소멸자에 의해 동일한 배열을 반환하게 되므로, 이때 실행 오류가 발생하여 비정상 종료된다.

6장 연습문제

이론 문제

1. ② 함수 중복

2. ③ 클래스의 멤버 함수는 중복할 수 없다.

3. ③ `double f(int x, int y);`
   ④ `int f(int x, int y);`

4. ① `void g();`
   ② `int g();`

5. ② 컴파일러에 의해 컴파일시

6. ② 컴파일러에 의해 컴파일 시

7. ④ `int x = add(3.14);`

8. ② `int x = add(3.14);`

9. ③ `void f3(int a, int b=0, int c);`

10. ② `int add(int a, int b); int add(int a, int &b);`

11. ④. 소멸자는 매개 변수를 가질 수 없다.

12. (1) `0`
    (2) `3`
    (3) `5hello`

13. 다음 호출문은 중복 함수 호출에 모호성을 가진다.

```
f(5);
```

이 호출문은 f(int a)를 호출하여 매개 변수 a에 5를 넘겨주는 것인지, f(int a, int b=0);를 호출하고 a에는 5를, b는 디폴트 값 0을 넘기고자 하는 것인지 모호하여 컴파일 오류가 발생한다. 그러므로 문제에 주어진 2개의 함수 중복은 모호성의 문제를 가지고 있다.

14. 다음 호출문은 중복 함수 호출에 모호성을 가진다.

```
area(5);
```

컴파일러는 5를 float로 변환하여 area(float f)를 호출할 것인지, double로 변환하여 area(double d)를 호출할 것인지 모호하여 컴파일 오류를 발생시킨다. 그러므로 문제에 주어진 2개의 함수 중복은 모호성의 문제를 가지고 있다.

15. ④. static 멤버 함수는 non-static 멤버 함수를 호출할 수 없다.

16. ③

7장 연습문제

이론 문제

1. ①

2. ④

3.

```
class Sample {
...
   friend SampleManager;
};
```

4.

```
class Sample {
...
   friend bool SampleManager::compare(Sample &a, Sample &b);
};
```

5. ● 컴파일 오류의 발생 위치와 오류의 원인

isValid() 함수에서 아래와 같이 Student 클래스의 private 멤버인 id를 사용하고 있기 때문에 오류가 발생한다.

```
bool isValid(Student s) {
   if(s.id > 0) return true; // 컴파일 오류. id는 private 멤버
   else return false;
}
```

● 오류의 바람직한 수정

이 오류를 해결하기 위해, id를 public 속성으로 변경해도 되지만, 멤버 변수는 private으로 유지하는 것이 바람직하므로, isValid() 함수를 Student 클래스의 프렌드로 선언한다.

```
class Student {
   int id;
```

```
public:
   Student(int id) { this->id = id;}
   friend bool isValid(Student s); // 프렌드 선언
};
```

6. ● 컴파일 오류의 발생 위치와 오류의 원인

show() 함수에서 아래와 같이 Student의 private 멤버인 id와 Professor의 private 멤버인 name을 사용하기 때문에 컴파일 오류가 발생한다.

```
void show(Student s, Professor p) {
   cout << s.id << p.name;
}
```

● 오류의 바람직한 수정

이 오류를 해결하기 위해, id와 name를 public 속성으로 변경해도 되지만, 멤버 변수는 private으로 유지하는 것이 바람직하므로, show() 함수를 Student와 Professor에 모두 프렌드로 삽입한다. 그러나 한 가지 show() 함수의 매개 변수 타입에 Student와 Professor가 모두 등장하기 때문에 전방 참조 문제(코드의 뒤쪽에서 선언되는 이름을 앞에서 미리 사용할 수 없는 문제)를 해결하기 위해서는 코드의 맨 앞에 Student와 Professor 이름을 선언해야 한다.

```
// show() 함수의 프렌드 선언에서 Student와 Professor를 사용하는 전방 참조 문제 해결을 위해 선언
class Student;
class Professor;

class Student {
   int id;
public:
   Student(int id) { this->id = id;}
   friend void show(Student s, Professor p);
};
class Professor {
   string name;
public:
   Professor(string name) { this->name = name;}
   friend void show(Student s, Professor p);
};
void show(Student s, Professor p) {
   cout << s.id << p.name;
}
```

7. ● 컴파일 오류의 발생 위치와 오류의 원인

오류의 발생 위치는 Person 클래스의 shopping() 함수 내의 다음 라인이다.

```
if(food.price < 1000)
```

price가 Food 클래스의 private 속성이므로 Person 클래스에서는 접근할 수 없기 때문이다.

● 오류의 바람직한 수정

이 오류를 해결하기 위해 Person 클래스의 shopping() 멤버 함수를 다음과 같이 Food 클래스에 프렌드로 선언하면 된다. 다만 전방 참조를 막기 위해 class Food;를 코드 맨 앞에 선언하고 shopping() 함수의 선언과 구현을 분리한다.

```
// Person 클래스에서 아직 선언되지 않는 Food 이름을 사용하는 전방 참조 문제를 해결하기 위해 다음 라인 선언
class Food;

class Person {
   int id;
public:
   void shopping(Food food);
   int get() { return id; }
};

class Food {
   int price;
   string name;
public:
   Food(string name, int price);
   void buy();
   friend void Person::shopping(Food food);
};

void Person::shopping(Food food) {
   if(food.price < 1000)
      food.buy();
}
```

8. ④

9. Sample 클래스의 friend bool isZero(Sample &a) { ... } 함수 선언은, isZero() 함수를 외부 함수로 선언하면서 동시에 프렌드로 선언하는 코드이다. isZero() 함수는 Sample 클래스의 멤버 함수가 아니므로 main()에서 다음과 같이 호출하면 컴파일 오류가 발생한다.

```
bool ret = a.isZero(b);
```

그리고 isZero()가 프렌드로서 의미가 있으려면, x 멤버를 private으로 선언하는 것이 바람직하다. 코드를 전체적으로 다음과 같이 수정한다.

```
class Sample {
   int x; // private으로 선언
public:
   Sample(int x) {this->x = x;}
   friend bool isZero(Sample &a) {
      if(a.x == 0) return true;
      else return false;
   }
};

int main() {
   Sample a(5), b(6);
   bool ret = isZero(b); // a.isZero(b);를 수정
}
```

10. Sample 클래스 내에 friend bool isZero(Sample &a);의 선언은 현재 필요하지 않다. isZero() 함수가 Sample 클래스의 멤버 x를 접근하는데 무리가 없기 때문이다. 만일 멤버 x를 private으로 선언하면 friend 선언은 꼭 필요하다.

11. << 연산자의 중복을 보여준다. 첫 라인의 코드에서는 C++의 기본 시프트 연산자(<<)의 본래 의미대로 사용한 사례이며, 두 번째 라인은 cout 스트림에 문자 'a'를 출력하도록 << 연산자가 중복 작성된 사례이다. << 연산자는 피연산자에 따라 서로 다른 의미로 실행되므로, 이것이 다형성이다.

12. 소주 + 맥주 = 소맥
탄소 + 산소 = 일산화탄소
돈 + 권력 = 정경유착

13. ④

14. ④

15. ③. `Power operator += (Power b);`이어야 한다.

16. ③. `Power operator ++ (Power& a, int b);`이어야 한다.

17. `Circle` 클래스의 경우 포인터를 가진 멤버가 없기 때문에, 두 객체를 비트 단위로 복사하는 객체의 기본 치환 연산(=)으로도 문제가 발생하지 않는다. 따로 치환 연산자를 작성할 필요가 없다.

8장 연습문제

이론 문제

1. ②

2. ①

3. ③

4. ④

5. ②

6. ①, ②, ④

7. (1) 업 캐스팅 ②, 다운 캐스팅 ③
   (2) q가 클래스 B의 포인터 타입이고 y가 클래스 B의 멤버이므로, `q->y = 100;`는 문법적으로는 문제가 없다. 그러나 실제 포인터 q가 가리키는 것은 A 타입의 객체이므로 q가 가리키는 객체 공간에는 멤버 y가 존재하지 않는다. `q->y = 100;`의 문장은 할당받지 않는 공간 y에 100을 쓰게 되므로 불법적인 메모리 접근이 되어 실행 중에 오류가 발생하고 비정상 종료할 수 있다.

8. (1) ③
   (2) z, x, w
   (3) ③
   (4) `dp = (D*)ap;`

9. (1) 생성자 A
   생성자 B
   (2) 생성자 A
   생성자 B 10
   (3) 생성자 A 32
   생성자 B 400

10. (1) 컴파일 오류가 발생하는 라인은 다음 두 라인이다.

```
B() { cout << "생성자 B" << endl; } // B() 부분에서 컴파일 오류
B(int x) { cout << "생성자 B " << x << endl; } // B(int x) 부분에서 컴파일 오류 발생
```

위의 두 생성자 B(), B(int x)가 컴파일러에 의해 묵시적으로 기본 클래스 A의 기본 생성자를 호출하도록 컴파일된다. 그러나 클래스 A에 기본 생성자가 선언되어 있지 않기 때문에, 컴파일 오류가 발생한다.

(2) 생성자 B()를 다음과 같이 수정한다.

```
B() : A(20) { cout << "생성자 B" << endl; }
```

(3) 생성자 B(int x)를 다음과 같이 수정한다.

```
B(int x) : A(x+20) { cout << "생성자 B " << x << endl; }
```

11. ③. 파생 클래스의 소멸자가 먼저 실행된 후 기본 클래스의 소멸자가 실행된다.

12. ④

13.

```
class Satellite : public Rocket, public Computer { };
```

14. (1)

```
class HiPen : public Pen, public Eraser { };
```

(2)

```
class OmniPen : public Pen, public Eraser, public Lock { };
```

15. 컴파일 오류가 발생하는 라인은 ④.
컴파일 오류의 원인은 FlyingCar가 Car와 Airplane을 다중 상속받아, power 멤버가 객체 fCar에 중복되어 생성되므로 fCar.power로 power 멤버를 접근할 때, 컴파일러는 중복된 power 중 어떤 것인지 모호하여 선택할 수 없기 때문이다.
컴파일 오류를 수정하기 위해 Car와 Airplane을 다음과 같이 가상 상속으로 선언한다.

```
class Car : virtual public Vehicle
class Airplane : virtual public Vehicle
```

16. 만일 새로운 클래스 SmartTV를 다음과 같이 선언하면,

```
class SmartTV : public ColorTV, public InternetTV {
};
```

다음과 같이 sTV 객체를 생성할 때, 다중 상속으로 인해 screenSize 멤버가 중복되어 생성되므로 screenSize 멤버에 접근할 때 모호성이 존재할 수 있다.

```
SmartTV sTV;
sTV.screenSize = 30; // screenSize 멤버에 대한 모호성 발생. 컴파일 오류
```

그러므로 문제에 주어진 클래스 ColorTV와 InternetTV를 가상 상속으로 다음과 같이 수정한다.

```
class ColorTV : virtual public TV
class InternetTV : virtual public TV
```

9장 연습문제

이론 문제

1. ① virtual

2. ② overriding

3. ③ compile-time binding

4. 다형성

5. (1) 기본 클래스는 Base, 파생 클래스는 Derived
   (2) Derived::f() called
   (3) Base::f() called
   (4) Base::f() called
   (5) Base::f() called

6. (1) ① A의 f() ② B의 f() ③ C의 f()
   (2) C::f() called
   (3) C::f() called
   (4) C::f() called
   (5) C::f() called

7. 동일한 이름의 변수나 함수가 여러 곳에 선언되었을 때, 가장 가까운 범위에 선언된 이름을 사용하는 규칙을 컴퓨터 언어 이론에서 범위 규칙(이)라고 한다. 범위 지정 연산자(을)를 사용하면 클래스 멤버와 동일한 이름의 외부 함수를 클래스 내에서 호출할 수 있다.

8. (1) ::f();
   (2) A::f();
   (3) f();

9. ② 소멸자를 가상 함수로 선언하는 것이 바람직하다.

10. (1) 실행 결과는 다음과 같다.

```
id=10
```

실행 결과의 문제점은 ~Student()가 실행되지 않는다는 점이다.

(2) Person 클래스의 소멸자를 다음과 같이 가상 소멸자로 수정하면 된다.

```
virtual ~Person() { cout << "id=" << id << endl; }
```

11. ③

12. ③

13. ②

14. (1) ① Shape shape; ③ Circle circle(10);

(2) Circle 클래스에 다음 멤버를 추가하면 된다.

```
void draw() { cout << "반지름=" << radius << "인 원" << endl; }
```

15. ④ 순수 가상 함수 호출

10장 연습문제

이론 문제

1. ③

2. ③

3. ②

4. ②

5.

```
template <class T>
bool equal(T a, T b) {
   if(a == b) return true;
   else return false;
}
```

6.

```
template <class T>
void insert(T a, T b[], int index) {
   b[index] = a;
}
```

7. max가 리턴하는 값은 T 타입의 값이므로 리턴 타입을 T로 해야 한다.

```
template <typename T> T max(T x, T y) {
   if(x > y) return x;
   else return y;
}
```

8. 매개 변수 y의 타입을 TYPE, 리턴 타입을 bool로 수정한다.

```
template <class TYPE>
bool equals(TYPE x, TYPE y) {
   if(x == y) return true;
   else return false;
}
```

9. (1)

```
int avg(int *p, int n) {
   int k;
   int sum=0;
   for(k=0; k<n; k++) sum += p[k];
   return sum/n;
}
```

(2)

```
double avg(double *p, int n) {
   int k;
   double sum=0;
   for(k=0; k<n; k++) sum += p[k];
   return sum/n;
}
```

10. (1) 예. 템플릿 함수와 보통 함수는 중복하여 공존할 수 있다.
(2) 3.14 (템플릿 함수를 호출하였음)
(3) special 100 (중복된 보통 함수를 호출하였음)

11. ③

12.

(1)

```
T* p; // T 타입의 포인터 p를 선언하라.
int size; // 배열에 현재 들어 있는 개수를 나타내는 변수 size를 선언하라.
```

(2)

```
template <class T> Container<T>::Container(int n) {
   p = new T [n];
   size = n;
}
```

(3)

```
template <class T> Container<T>::~Container() {
   delete [] p;
}
```

(4)

```
template <class T> T Container<T>::get(int index) {
   return p[index];
}
```

(5)

```
int main() {
   Container<char> c(26);
}
```

(6)

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

int main() {
   Container<char> c(26);
   for(int i=0; i<26; i++)
      c.set(i, 'a'+i);
   for(int i=25; i>=0; i--)
      cout << c.get(i);
}
```

13. ①

14. (1) vector 클래스 --> <vector>
(2) list 클래스 --> <list>
(3) merge 함수 --> <algorithm>
(4) search 함수 --> <algorithm>

15.

```
vector<double> v; // double 타입의 벡터 v 생성
v.push_back(3.1);
v.push_back(4.1);

// 벡터 v의 모든 값을 출력하라.
for(int i=0; i<v.size(); i++)
   cout << v[i] << endl;
```

16.

```
void print(vector<char> &v) {
   vector<char>::iterator it;
   for(it = v.begin(); it!=v.end(); it++) {
      char c = *it;
      cout << c;
   }
}
```

11장 연습문제

이론 문제

1. ③

2. ④

3. 키보드

4. 스크린

5. cin, cout, cerr, clog

6. ② istream

7. ①

8. char

9. 실행 결과 화면에 abc가 출력된다. 이유는 put()은 ostream& 타입으로 cout(현재 객체)의 참조를 리턴하므로, cout.put('a')에서 출력된 'a'는 cout 스트림에 삽입되고, cout.put('a').put('b')는 다시 'b'를 cout의 버퍼에 삽입한다. 그러므로 모두 cout과 연결된 스크린 장치에 출력된다. 그러나 만일 put() 함수의 리턴 타입이 ostream이라면, cout.put('a')는 cout 버퍼에 'a'를 쓰고 cout의 복사본을 리턴한다. 그리고 cout.put('a').put('b')는 'b'를 cout이 아닌 cout의 복사본 스트림에 쓰게 되어 어떤 결과가 나타날지 예측할 수 없다.

10. get()과 getline() 모두 '\n'(<Enter> 키)을 만날 때까지 최대 99개의 문자를 입력받고, 마지막에 '\0' 을 사용자 버퍼에 추가하는 것은 동일하다. 그러나 get()은 키보드로부터 읽은 '\n'을 입력 스트림 버퍼에 그대로 남겨두지만, getline()은 '\n'을 입력 스트림 버퍼에 남겨두지 않고 제거한다.

11. ①

12. 15. getline()은 '\n'을 읽고 스트림 버퍼에 남겨 두지 않고 제거하므로 <Enter> 키도 읽은 문자의 개수에 포함시킨다.

13. ②, ③

14. ④

15. ④

16. 출력 결과는 다음과 같다.
```
%%%%%%%%C++
```

17. 출력 결과는 다음과 같다.
```
0.6667~~~~
```

18. <iostream>과 <iomanip>

19. <ostream> 헤더 파일, 원형은 ostream& operator << (char);

20. ④

21. istream& ignoredigit(istream&);

22. ostream& ten(ostream&);

12장 연습문제

이론 문제

1. ① test.hwp

2. ④ iostream

3. <fstream>

4. ④

5. (1) 12바이트
   (2) 다음과 같은 12개의 16진수 값들이 들어 있다.
   0x57, 0x65, 0x6C, 0x63, 0x6F, 0x6D, 0x65, 0x0D, 0x0A, 0x43, 0x2B, 0x2B
   (3) 11. get()이 리턴한 문자는 Welcome + \n + C++로 총 11개 카운트 됨
   (4) 12. 바이너리 I/O로 읽으면 Welcome + \r + \n + C++로 총 12개 카운트 됨

6. (1) 19바이트
   (2) 다음과 같은 19개의 16진수 값들이 들어 있다.
   0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x0D 0x0A
   0x49 0x6E 0x74 0x65 0x67 0x65 0x72
   (3) 18. get()이 리턴한 문자는 0123456789 + \n + Integer로 총 18개 카운트 됨
   (4) 19. 바이너리 I/O로 읽으면 0123456789 + \r + \n + Integer의 총 19개 카운트 됨. count 값이 서로 다른 이유는 binary I/O 모드를 사용하면 파일 속에 있는 '\r'도 하나의 문자로 읽기 때문

7. ifstream fin("test.txt");

8. if 문은 다음과 같다.

```
if(!fout) {
   cout << "열기 실패";
   return 0;
}
```

9. ③

10. ②

11. 256바이트 단위로 텍스트 파일을 읽고 화면에 출력하는 완성된 코드이다.

```
#include <iostream>
#include <fstream>
using namespace std;
void fread(ifstream &fin) {
   char buf[256]; // 버퍼 buf를 선언한다.
   while(!fin.eof())
      fin.read(buf, 256);
      int n = fin.gcount(); // 실제 읽은 바이트 수를 알아낸다.
      cout.write(buf, n); // write()를 이용하여 읽은 데이터를 화면에 출력한다.
   }
}

int main() {
   ifstream fin("c:\\windows\\system.ini");
   fread(fin);
}
```

12. 1024바이트 단위로 텍스트 파일을 읽고 화면에 출력하는 완성된 코드이다.

```
#include <iostream>
#include <fstream>
using namespace std;

void fread(ifstream &fin) {
   char buf[1024]; // 버퍼 buf를 선언한다.
   while(true) {
      fin.read(buf, 1024);
      int n = fin.gcount(); // 실제 읽은 바이트 수를 알아낸다.
      cout.write(buf, n); // write()를 이용하여 읽은 데이터를 화면에 출력한다.
      if(n < 1024) // 읽은 바이트가 1024보다 작으면
         break; // 루프를 중단한다.
   }
}

int main() {
   ifstream fin("c:\\windows\\system.ini");
   fread(fin);
}
```

13. EOF(-1) 데이터는 파일에 존재하지 않는다. 운영체제가 파일의 끝을 인식하여, 입출력 함수가 운영체제 API를 호출하면 알려준다.

14. ③

15. ④

16. ②

17. 바이너리 파일에 0x0D 0x0A 값이 연속하여 들어 있을 때, 텍스트 I/O 모드로 읽으면 0x0D 값을 생략하고 0x0A 만 리턴하여 문제가 생긴다.

18. 문제가 없다. 텍스트 파일의 라인 끝에는 '\r', '\n'이 연속적으로 들어 있으며, 바이너리 I/O 모드로 읽으면 텍스트 I/O와는 달리 '\r'을 읽어 올 수 있기 때문이다. 바이너리 I/O의 경우 '\r' 문자로 라인 끝을 판별하는 데는 문제가 없다.

19. ②

20. fin.seekg(100, ios::beg); 혹은 fin.seekg(100);

21. 30

22. length-i-1

13장 연습문제

이론 문제

1. ① 예외

2. ③ except

3. ③. catch () { } 블록은 여러 개 만들 수 있으며, 하나의 try { } 블록에는 최소한 하나의 catch() { } 블록이 있어야 한다.

4. ③. catch() { } 블록의 예외가 처리된 다음 catch() { } 블록 아래의 정상적인 코드로 실행이 계속된다.

5. 100

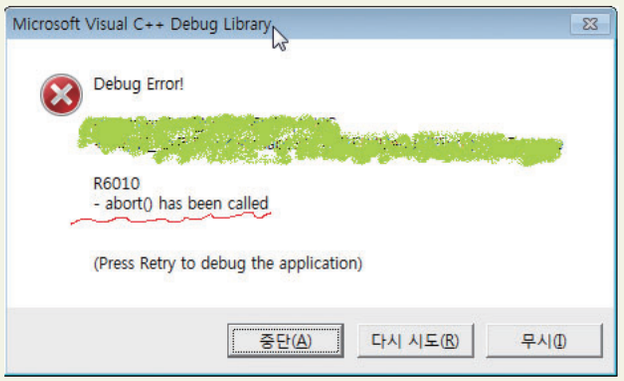
6. 3aa

7. 4

8. (1) 0을 다루지 않음
   (2) 0
   (3) 4

9. (1) "10"이 화면에 출력됨
   (2) "음수를 다루지 않음-3"이 화면에 출력됨

10. throw n; 문이 try { } 블록 안에 있지 않기 때문에, throw n; 문장이 실행될 때, 시스템에 의해 강제 종료된다. 이때 다음과 같은 오류 메시지가 출력된다.



11. 코드는 다음과 같다.

```
int get() throw(char*) {
   int n;
   cout << "0에서 10까지의 수를 입력>>";
   cin >> n;
   if(n < 0)
      throw "음수 입력 불가";
   if(n > 10)
      throw "너무 큰 숫자";
   return n;
}
```

12. int 타입을 리턴하는 함수 big()은 내부에 int 타입의 예외를 던지는 throw 문을 가지고 있으며, 상황에 따라 int 타입의 예외를 던진다. 그러므로 big() 함수를 호출하는 코드는 다음과 같이 try-catch 블록으로 묶어 예외를 처리할 수 있도록 해야 한다.

```
try {
   ...
   big()
   ...
}
catch(int x) {
   ...
}
```

13. ②

14. isEven() 함수에는 char* 타입의 예외를 던지는 코드가 들어 있으므로 isEven() 함수 선언문을 다음과 같이 수정해야 한다.

```
bool isEven(int x) throw(char *)
```

15. 이름 규칙, naming mangling

16. put() 함수는 C++ 함수이므로 목적 코드를 만들 때 C++ 이름 규칙을 적용하고, print() 함수는 C 언어로 작성되었기 때문에 목적 코드를 만들 때 C 이름 규칙을 적용할 것을 컴파일러에게 지시한다. 또한 put() 함수는 다른 파일에 작성된 함수 임을 선언한다.

17. C 언어는 컴파일할 때 함수의 매개 변수 타입이나 개수 등을 고려하지 않고 함수 본래의 이름에 _를 붙이는 이름 규칙을 사용하기 때문에, 소스 파일에 같은 이름으로 중복 작성된 함수들은 컴파일 후 모두 동일한 이름으로 목적 코드가 생성되기 때문이다.

18. _print, _main

19. add()는 C 소스파일에 작성된 C 함수이다. 그러므로 C++로 작성된 main.cpp의 main() 함수의 다음 코드에서 링크 시에 오류가 발생한다.

```
add(3,5);
```

링크 오류를 수정하려면 다음과 같이 main.cpp를 수정한다.

```
extern "C" int add(int x, int y);
int main() }
   int n = add(3,5);
}
```

20. (1) subtract.c의 컴파일 시에 오류가 발생하지 않는다.
(2) multiply.c의 컴파일 시에 오류가 발생하지 않는다.
(3) main.cpp의 컴파일 시에 오류가 발생하지 않는다.
(4) 링크할 때 오류가 발생한다. 링크 오류를 수정하려면 main.cpp의 함수 선언문을 다음과 같이 수정해야 한다.

```
extern "C" {
   int subtract(int x, int y);
   int multiply(int x, int y);
}
```