객체지향언어(11)

6주차 연습문제 과제

제출일: 2023.10.15

전공: 게임공학과

학번: 2019184020

성명: 윤은지

목차

1. **Rect의 객체를 다루는 다음 코드를 작성하려고 한다. 아래의 문제에 따라 빈칸에 적절한 코드를 삽입하라.**

아래는 주어진 코드에서 주석에 따라 포인터 변수를 선언하고 이를 이용하여 객체 'r'의 폭과 높이를 출력하는 방법입니다.

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

class Rect {
        int width, height;
public:
        Rect(int w, int h) { width = w; height = h; }
        int getWidth() { return width; }
        int getHeight() { return height; }
        int getArea();
};

int Rect::getArea() {
        return width * height;
}

int main()
{
        Rect r(2, 3);
        //(1) Rect 클래스에 대한 포인터 변수 p를 선언하라.
    Rect* p; // (1) 포인터 변수 p를 선언

        //(2) 선언된 포인터 변수 p에 객체 r의 주소를 지정하라.
    p = &r; // (2) 포인터 변수 p에 객체 r의 주소를 할당

        //(3) 포인터 변수 p를 이용하여 객체 r의 폭과 높이를 출력하라.
```

```
    cout << "폭: " << p->getWidth() << endl;
    cout << "높이: " << p->getHeight() << endl;
}
```

위의 코드에서 '(1)'에서는 'Rect' 클래스에 대한 포인터 변수 'p'를 선언하고, '(2)'에서는 'p'에 객체 'r'의 주소를 할당합니다. 그런 다음 '(3)'에서는 포인터 변수 'p'를 사용하여 객체 'r'의 폭과 높이를 출력합니다. 이렇게 하면 포인터를 사용하여 객체의 멤버 변수에 접근할 수 있습니다.

2. **사용자로부터 폭과 높이 값을 입력받아 동적으로 Rect 객체를 생성하고 면적을 구하여 출력하는 코드를 작성하고자 한다. 다음 물음에 따라 빈칸을 채워라.**

다음과 같이 코드를 완성할 수 있습니다.

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

class Rect {
    int width, height;
public:
    Rect(int w, int h) { width = w; height = h; }
    int getWidth() { return width; }
    int getHeight() { return height; }
    int getArea();
};

int Rect::getArea() {
    return width * height;
}

int main()
{
```

```
    Rect* q;
    int w, h;
    cin >> w >> h; // 사용자로부터 사각형의 폭과 높이를 w, h에 각각 입력받는다.

    //(1) 포인터 변수 q에 wxh 크기의 사각형을 표현하는 Rect 객체를 동적으로 생성한다.
    q = new Rect(w, h);

    //(2) 포인터 q를 이용하여 사각형의 면적을 출력한다.
    cout << "사각형의 면적: " << q->getArea() << endl;

    //(3) 생성한 객체를 반환한다.
    delete q;
}
```

위의 코드에서

- '(1)'에서는 'new'연산자를 사용하여 'Rect' 클래스의 동적 객체를 생성하고, 포인터 변수 'q'에 할당합니다.
- '(2)'에서는 포인터 'q'를 사용하여 사각형의 면적을 출력합니다.
- '(3)'에서는 동적으로 할당한 객체를 반환하기 전에 'delete' 연산자를 사용하여 메모리를 해제합니다. 이것은 동적 할당된 객체의 메모리 누수를 방지하기 위한 중요한 단계입니다.

3. Rect 객체나 배열을 생성하는 다음 코드 중 컴파일 오류가 발생하는 것은?

주어진 코드 중 컴파일 오류가 발생하는 경우는 다음과 같습니다.

1) 'Rect a;' 이 코드에서는 컴파일 오류가 발생하지 않습니다. 'Rect' 클래스의 기본 생성자가 호출되어 객체 'a'가 생성됩니다.
2) 'Rect b(5, 6);' 이 코드에서는 컴파일 오류가 발생하지 않습니다. 'Rect' 클래스의 매개변수를 받는 생성자가 호출되어 객체 'b'가 폭 5, 높이 6으로 생성됩니다.
3) 'Rect c[2] = {Rect(1,1), Rect(2,3)};' 이 코드에서 컴파일 오류가 발생하지 않습니다. 배열 'c'는 두 개의 'Rect' 객체로 초기화 되며, 각 객체는 주어진 폭과 높이로 생성됩니다.
4) 'Rect d[2][3] = {Rect(1,2), Rect(2,3), Rect(3,4), Rect(1,1), Rect(2,2), Rect(3,3)};'

이 코드에서 컴파일 오류가 발생합니다. 이 코드는 2x3 크기의 'Rect' 객체 다차원 배열을 초기화하려는 시도인데, 배열의 크기와 초기화 값의 개수가 일치하지 않습니다. 'Rect' 배열의 크기를 맞춰주어야 합니다.

4. Rect 객체의 배열을 생성하는 다음 코드는 컴파일 오류가 발생한다. 컴파일 오류가 발생하지 않기 위해 Rect 클래스를 어떻게 수정하여야 하는가?

주어진 코드에서 컴파일 오류가 발생하는 이유는 'Rect' 클래스가 기본 생성자를 정의하지 않았기 때문입니다. C++에서, 동적 배열을 생성할 때 기본 생성자를 호출하여 요소를 초기화해야 합니다.

따라서, 'Rect' 클래스에 기본 생성자를 추가해야 합니다. 기본 생성자는 매개변수 없이 객체를 초기화하는 역할을 합니다.

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

class Rect {
  int width, height;
public:
  Rect() : width(0), height(0) {} // 기본 생성자 추가
  Rect(int w, int h) : width(w), height(h) {}

  int getWidth() { return width; }
  int getHeight() { return height; }
  int getArea();
};

int Rect::getArea() {
  return width * height;
}

int main()
{
```

```
    Rect* p = new Rect[10]; // 이제 컴파일 오류가 발생하지 않습니다.

    // 배열을 사용한 예시
    for (int i = 0; i < 10; i++) {
        cout << "Rect " << i + 1 << ": Width = " << p[i].getWidth() << ", Height = " << p[i].getHeight() << ", Area = " << p[i].getArea() << endl;
    }

    delete[] p; // 메모리를 해제
}
```

위의 코드에서 'Rect' 클래스에 기본 생성자가 추가되었고, 이제 'new Rect[10]'를 사용하여 'Rect' 객체 배열을 동적으로 생성하는 것이 가능해집니다.

5. Rect 클래스에 다음과 같은 기본 생성자를 삽입하고, 다음 배열 r 생성 후, 배열 r의 사각형 면적의 합을 출력하는 코드를 작성하라.

주어진 'Rect' 클래스에 기본 생성자를 삽입하고, 배열 'r'을 사용하여 사각형 면적의 합을 출력하는 코드는 다음과 같이 작성할 수 있습니다.

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

class Rect {
    int width, height;
public:
    Rect() { width = 1; height = 1; } // 기본 생성자 추가
    Rect(int w, int h) : width(w), height(h) {}

    int getWidth() { return width; }
    int getHeight() { return height; }
    int getArea();
```

```
};

int Rect::getArea() {
    return width * height;
}

int main()
{
    Rect r[5] = {Rect(), Rect(2, 3), Rect(3, 4), Rect(4, 5), Rect(5, 6)};

    int totalArea = 0;
    for (int i = 0; i < 5; i++) {
        cout << "Rect " << i + 1 << ": Width = " << r[i].getWidth() << ", Height = " << r[i].getHeight() << ", Area = " << r[i].getArea() << endl;
        totalArea += r[i].getArea();
    }

    cout << "Total Area: " << totalArea << endl;
}
```

위 코드에서 'Rect' 클래스에 기본 생성자가 추가되었고, 배열 'r'을 사용하여 각 사각형의 면적을 계산하고 출력한 후, 모든 사각형의 면적의 합을 출력합니다.

6. public 속성의 getVolume() 멤버 함수를 가진 Cube 클래스에 대해, 다음 코드가 있다. 다음 중 컴파일 오류가 발생하는 것은?

주어진 코드에서 'Cube' 클래스의 객체 'c'와 포인터 'p'에 대해 'getVolume()'함수를 호출하는 방식은 다음과 같습니다.

1) 'c.getVolume();' 이것은 객체 'c'의 'getVolume()' 멤버 함수를 호출하는 올바른 방법입니다.
2) 'p->getVolume();' 이것은 포인터 'p'를 사용하여 객체의 멤버 함수를 호출하는 올바른 방법입니다.
3) '(*p).getVolume();' 이것 또한 포인터 'p'를 사용하여 객체의 멤버 함수를 호출하는 올바

른 방법입니다.

4) ‘c->getVolume();’ 이 옵션이 컴파일 오류를 발생시킵니다. ‘c’는 ‘Cube’ 클래스의 객체이지만 포인터가 아니기 때문에 화살표 연산자 (‘->’)를 사용할 수 없습니다. 포인터에 사용되는 ‘->’ 연산자는 객체가 아닌 포인터 변수에 사용됩니다.

따라서, 올바른 호출 방법은 1), 2), 3)입니다. 옵션 4)는 컴파일 오류를 발생시킵니다.

7. 다음 객체 배열에 관해 잘못 설명된 것은?

다음 객체 배열에 관한 설명은 다음과 같이 수정되어야 합니다.

4) ‘delete c;’ 코드로 배열 ‘c’를 소멸하는 것은 잘못된 방법입니다. 배열을 소멸할 때 ‘delete[]’ 연산자를 사용해야 합니다. 정확한 코드는 ‘delete[] c;’입니다. (잘못된 설명)

8. 다음 프로그램이 실행될 때 출력되는 결과는 무엇인가?

프로그램이 실행되면 ‘Palette’ 클래스의 기본 생성자가 호출됩니다. ‘Palette’ 클래스의 생성자에서 ‘Color’ 객체 배열 ‘p’가 동적으로 생성됩니다. 배열 ‘p’의 크기는 3이며, 각 요소는 ‘Color’ 클래스의 기본 생성자를 호출합니다. 따라서 ‘Color’ 클래스의 기본 생성자가 3번 호출되어 “기본생성자”가 3번 출력됩니다.

그런 다음 ‘Palette’ 클래스의 소멸자가 호출됩니다. 소멸자에서는 ‘p’가 메모리에서 해제됩니다. 이 과정에서 ‘Color’ 객체들도 소멸자가 호출됩니다. 따라서 “소멸자”가 호출되는 메시지가 출력되지만, “매개변수생성자” 메시지는 출력되지 않습니다.

따라서 프로그램이 실행될 때 출력되는 결과는 “기본생성자”가 3번, “소멸자”가 3번입니다.

9. New와 delete는 무엇인가?

‘new’와 ‘delete’는 C++에서 사용되는 연산자로서 다음과 같습니다.

3) C++의 표준 객체

‘new’ 연산자는 동적으로 메모리를 할당하여 객체를 생성하는 데 사용되며, ‘delete’ 연산자는 동적으로 할당된 메모리를 해제하는 데 사용됩니다. 이러한 연산자들은 객체 지향 프로그래밍에서 메모리 관리와 객체 생성과 파괴를 위한 중요한 도구로 사용됩니다.

10. 다음 코드의 문제점은 무엇인가?

주어진 코드에는 다음과 같은 문제점이 있습니다.

'new Cube [4]'를 사용하여 Cube 클래스의 동적 배열을 생성하였으므로, 해당 배열을 해제할 때 'delete[]'를 사용해야 합니다.

'delete'를 사용하면 메모리 누수와 런타임 오류가 발생할 수 있습니다. 올바른 방법은 'delete[] p;'입니다.

수정된 코드는 다음과 같아야 합니다.

```
Cube* p = new Cube[4];

delete[] p;
```

이렇게 수정하면 메모리가 올바르게 해제되고, 프로그램이 올바르게 실행됩니다.

11. This에 대해 잘못 말한 것은?

This 포인터에 대한 설명 중에서 잘못된 것은 다음과 같습니다.

3) this는 static 함수를 포함하여 멤버 함수 내에서만 다루어지는 객체 자신에 대한 포인터이다. (잘못된 설명)

설명 3이 잘못되었습니다. This 포인터는 멤버 함수 내에서 객체 자신을 가리키는 포인터입니다. 객체가 정적(static)인지 아니면 인스턴스(객체)인지에 관계없이 this 포인터를 사용할 수 있습니다.

12. This의 활용에 대해 잘못 설명한 것은?

2) This는 static 멤버 함수에는 사용할 수 없다. (잘못된 설명)

This 포인터는 일반 멤버 함수 내에서 사용되며, 해당 멤버 함수가 속한 객체의 주소를 가리킵니다. 그러나 static 멤버 함수는 특정 객체 인스턴스에 묶이지 않으므로 this 포인터를 사용할 수 없습니다. This 포인터는 동적으로 생성된 객체를 가리키는 역할을 합니다.

13. This를 최대한 많이 활용하여 다음 클래스를 가장 바람직하게 수정하라.

'this' 포인터를 최대한 활용하여 'Location' 클래스를 수정하려면 다음과 같이 클래스 정의를 변경할 수 있습니다.

```
#include <iostream>

using namespace std;


class Location {
        int width, height;
public:
        Location() : width(0), height(0) { }
        Location(int w, int h) : width(w), height(h) { }
        void show();
        Location& setWidth(int w);
        Location& setHeight(int h);
        Location& setDimensions(int w, int h);
};


void Location::show() {
        cout << "Width: " << width << ", Height: " << height << endl;
}


Location& Location::setWidth(int w) {
        this->width = w;
        return *this;
}


Location& Location::setHeight(int h) {
        this->height = h;
        return *this;
}
```

```
Location& Location::setDimensions(int w, int h) {
        this->setWidth(w);
        this->setHeight(h);
        return *this;
}

int main() {
        Location loc;
        loc.show();

        loc.setWidth(5).setHeight(10).show();

        loc.setDimensions(8, 12).show();
}
```

위의 수정된 코드에서 'Location' 클래스는 'this' 포인터를 사용하여 멤버 함수를 체이닝하고, 객체의 속성을 설정하고 출력하는 방법을 제공합니다. 이를 통해 객체를 초기화하고 속성을 설정하는 과정이 더 효과적이고 가독성이 좋아집니다.

14. 메모리 누수란 어떤 상황에서 발생하는가?

메모리 누수는 주로 프로그램이 동적으로 메모리를 할당하고 그 메모리를 해제하지 않을 때 발생합니다.

15. 함수 f()가 실행되고 난 뒤 메모리 누수가 발생하는지 판단하고 메모리 누수가 발생하면 발생하지 않도록 수정하라.

(1)

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

```
    delete[] p; // 동적으로 할당된 메모리를 해제
}
```

이 함수에서 메모리 누수가 발생합니다. 메모리 할당 후 해제가 이루어지지 않았기 때문에 'p'가 가리키는 메모리는 해제되지 않고 남게 됩니다. 따라서 'delete[] p;'를 사용하여 메모리를 해제해야 합니다.

(4)

```
void f() {
    int* p;
    for (int i = 0; i < 5; i++) {
        p = new int;
        cin >> *p;
        if (*p % 2 == 1) break;
        delete p; // 루프 내에서 할당한 메모리를 각각 해제
    }
    delete p; // 마지막 할당한 메모리를 해제
}
```

이 함수에서 메모리 누수가 발생합니다. 루프 내에서 할당한 메모리는 해당 루프 내에서 해제되어야 합니다. 따라서 루프에서 'delete p;'를 사용하여 메모리를 해제해야 합니다. 또한, 루프 종료 후에도 'p'가 가리키는 메모리가 할당되었으므로 마지막에 'delete p;'를 사용하여 메모리를 해제해야 합니다.