C++ 퀴즈 (2023-01-03)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 다음 코드의 출력 값과 그 이유는?

출력 값: 다르다

이유: a == b는 a배열과 b배열의 0번지 주소 값을 비교하는 것이므로 다르다.

텍스트이(가) 표시된 사진

자동 생성된 설명

번외 문제. 다음 코드의 출력 값과 그 이유는?

출력 값: 다르다

이유: “abcd”가 저장되어 있는 메모리의 주소를 저장하는 것이므로 주소 값을 비교하는 것이므로 다르다라고 생각하는데 컴파일 해보니 같다라고 나와서 이유를 모르겠다

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 다음 코드가 컴파일되지 않는 이유와 해결 방법을 서술하시오.

이유: Derived는 Base를 public으로 상속 받고 있는데 private 변수의 접근하였기 때문이다. Base의 string s를 protected 멤버 변수로 바꾸면 해결된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 다음 코드의 출력 값과 그 이유는?

출력 값: derived class

이유: 변수 d에서 printSome 함수를 호출하면 상속받고 있는 base 클래스에도 같은 이름의 함수가 있으므로 함수 오버라이딩이 발생하여 근처에 있는 Derived 클래스의 printSome을 호출하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 블로그에선 직원 데이터와 매니저 데이터를 관리하기 위해 이중 포인터를 선언하였고, 이를 new를 통해 동적 할당하여 Employee\*의 배열과 Manager\*의 배열을 생성하였다. 그리고 이 배열을 다시 동적 할당하여 동적 할당된 지역에 Employee와 Manager를 선언하였는데, 반드시 이럴 필요가 있을지 생각해 보자.

다시 말하자면, Employee\*만 선언하여 먼저 배열을 동적 할당하고, 여기에 Employee를 대입하는 방법으로 블로그에서 구현했던 기능을 완전히 같게 구현할 수 있지 않을까?

Employee\*만 선언하여 구현

#include <iostream>

#include <string>

class Employee {

protected:

std::string name;

int age;

std::string position; // 직책 (이름)

int rank; // 순위 (값이 클 수록 높은 순위)

public:

Employee(std::string name, int age, std::string position, int rank)

: name(name), age(age), position(position), rank(rank) {}

// 복사 생성자

Employee(const Employee& employee) {

name = employee.name;

age = employee.age;

position = employee.position;

rank = employee.rank;

}

// 디폴트 생성자

Employee() {}

void print\_info() {

std::cout << name << " (" << position << " , " << age << ") ==> "

<< calculate\_pay() << "만원" << std::endl;

}

int calculate\_pay() { return 200 + rank \* 50; }

};

class Manager : public Employee {

int year\_of\_service;

public:

Manager(std::string name, int age, std::string position, int rank,

int year\_of\_service)

: Employee(name, age, position, rank), year\_of\_service(year\_of\_service) {}

// 복사 생성자

Manager(const Manager& manager)

: Employee(manager.name, manager.age, manager.position, manager.rank) {

year\_of\_service = manager.year\_of\_service;

}

// 디폴트 생성자

Manager() : Employee() {}

int calculate\_pay() { return 200 + rank \* 50 + 5 \* year\_of\_service; }

void print\_info() {

std::cout << name << " (" << position << " , " << age << ", "

<< year\_of\_service << "년차) ==> " << calculate\_pay() << "만원"

<< std::endl;

}

};

class EmployeeList {

int alloc\_employee; // 할당한 총 직원 수

int current\_employee; // 현재 직원 수

int current\_manager; // 현재 매니저 수

Employee\* employee\_list; // 직원 데이터

Manager\* manager\_list; // 매니저 데이터

public:

EmployeeList(int alloc\_employee) : alloc\_employee(alloc\_employee) {

employee\_list = new Employee [alloc\_employee];

manager\_list = new Manager [alloc\_employee];

current\_employee = 0;

current\_manager = 0;

}

void add\_employee(Employee employee) {

// 사실 current\_employee 보다 alloc\_employee 가 더

// 많아지는 경우 반드시 재할당을 해야 하지만, 여기서는

// 최대한 단순하게 생각해서 alloc\_employee 는

// 언제나 current\_employee 보다 크다고 생각한다.

// (즉 할당된 크기는 현재 총 직원수 보다 많음)

employee\_list[current\_employee] = employee;

current\_employee++;

}

void add\_manager(Manager manager) {

manager\_list[current\_manager] = manager;

current\_manager++;

}

int current\_employee\_num() { return current\_employee + current\_manager; }

void print\_employee\_info() {

int total\_pay = 0;

for (int i = 0; i < current\_employee; i++) {

employee\_list[i].print\_info();

total\_pay += employee\_list[i].calculate\_pay();

}

for (int i = 0; i < current\_manager; i++) {

manager\_list[i].print\_info();

total\_pay += manager\_list[i].calculate\_pay();

}

std::cout << "총 비용 : " << total\_pay << "만원 " << std::endl;

}

~EmployeeList() {

delete[] employee\_list;

delete[] manager\_list;

}

};

int main() {

EmployeeList emp\_list(10);

emp\_list.add\_employee(Employee("노홍철", 34, "평사원", 1));

emp\_list.add\_employee(Employee("하하", 34, "평사원", 1));

emp\_list.add\_manager(Manager("유재석", 41, "부장", 7, 12));

emp\_list.add\_manager(Manager("정준하", 43, "과장", 4, 15));

emp\_list.add\_manager(Manager("박명수", 43, "차장", 5, 13));

emp\_list.add\_employee(Employee("정형돈", 36, "대리", 2));

emp\_list.add\_employee(Employee("길", 36, "인턴", -2));

emp\_list.print\_employee\_info();

return 0;

}

이런 식으로 구현하면 직원 수의 크기만큼 할당 받아야 하므로 직원수가 많아지면 많아질수록 메모리가 많이 사용된다고 생각이든다.

이중 포인터를 사용하면 할당한 직원 수의 크기가 포인터 배열로 8byte로 고정되어 있으므로 할당한 직원 수의 크기가 8byte보다 작은 것이 아니라면 이중 포인터가 효율적이라고 생각한다.