REPORT



|  |  |
| --- | --- |
| 과 목 : | 객체지향프로그래밍 |
| 제출일자 : | 2023.05.02 |
| 담당교수 : | 황성호 |
| 학 과 : | 멀티디자인학과 |
| 학 번 : | 201522405 |
| 이 름 : | 최준하 |

1.(예제 5-5) 참조 매개 변수로 평균 리턴하기.

#include <iostream>

using namespace std;

bool average(int a[], int size, int& avg) {

if (size <= 0) {

return false;

}

int sum = 0;

for (int i = 0; i < size; i++)

{

sum += a[i];

}

avg = sum / size; //참조 매개변수

return true;

}

int main() {

int x[] = { 0,1,2,3,4,5 };

int avg;

if (average(x, 6, avg)) {

cout << "평균은" << avg << endl;

}

else

{

cout << "매개 변수 오류" << endl;

}

if (average(x, -2, avg)) {

cout << "평균은" << avg << endl;

}

else

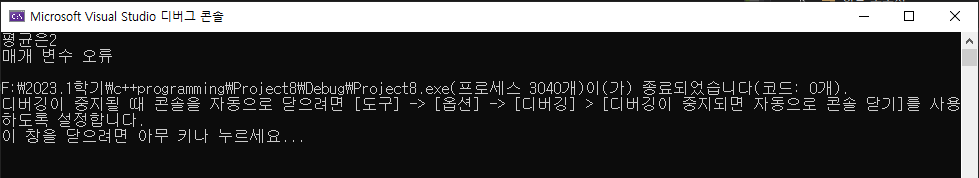
{

cout << "매개 변수 오류" << endl;

}

}

🡪출력 결과



2. (예제 5-8) 참조 리턴.

#include <iostream>

using namespace std;

char& find(char s[], int index) {

return s[index];

}

int main() {

char name[] = "mike";

cout << name << endl;

find(name, 0) = 's';

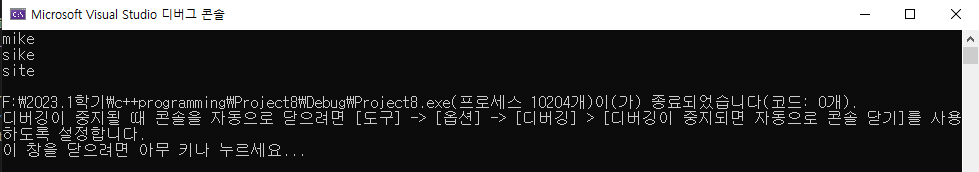
cout << name << endl;

char& ref = find(name, 2);

ref = 't';

cout << name << endl;

🡪출력 결과



3. (예제 5-9) circle 클래스의 복사 생성자와 객체 복사.

#include <iostream>

using namespace std;

class \_circle {

private:

int radius;

public:

\_circle(const \_circle& c); // 복사 생성자

\_circle() { radius = 1; }

\_circle(int radius) { this->radius = radius; }

double get\_area() { return 3.14 \* radius \* radius; }

};

\_circle::\_circle(const \_circle& c) {

this->radius = c.radius;

cout << "복사생성자 실행 radius = " << radius << endl;

}

int main() {

\_circle src(30); //보통

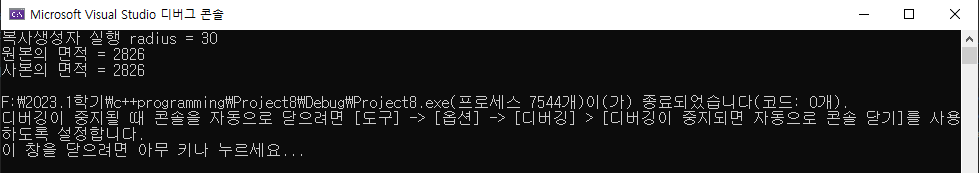
\_circle dest(src); //복사

cout << "원본의 면적 = " << src.get\_area() << endl;

cout << "사본의 면적 = " << dest.get\_area() << endl;

}

🡪출력 결과



4. (예제 5-10) 얕은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우.

#pragma warning(disable:4996)

#include <iostream>

#include <cstring>

using namespace std;

class \_person {

char\* name;

int id;

public:

\_person(int id, const char\* name); //생성자 컴파일러 복사 자동생성

~\_person();

void change\_name(const char\* name);

void show() { cout << id << ',' << name << endl; }

};

\_person::\_person(int id, const char\* name) {

this->id = id;

int len = strlen(name); //문자개수

this->name = new char[len + 1];

strcpy(this->name, name); //문자열 복사

}

\_person::~\_person() {

if (name) {

delete[] name;

}

}

void \_person::change\_name(const char\* name) {

if (strlen(name) > strlen(this->name)) { return; }

strcpy(this->name, name);

}

int main() {

\_person father(1, "kitae");

\_person daughter(father);

cout << "daughter 객체 생성 직후 ---" << endl;

father.show();

daughter.show();

daughter.change\_name("grace");

cout << "duaghter 이름을 grace로 변경한 후 ---" << endl;

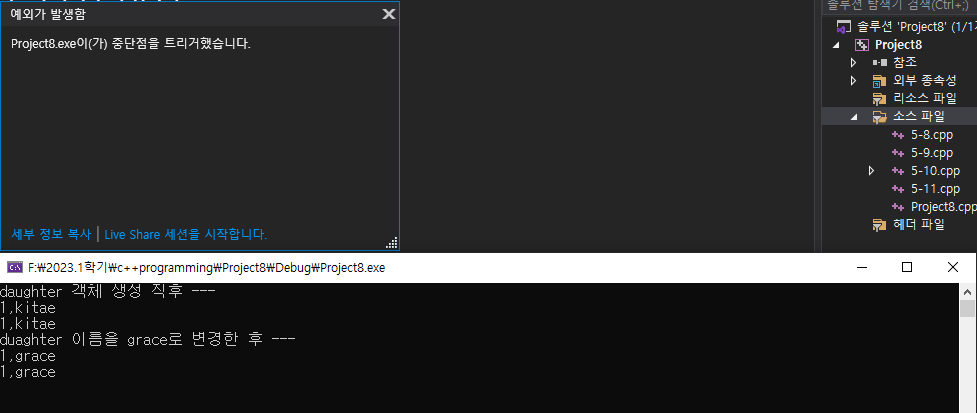
father.show();

daughter.show();

return 0;

}

🡪출력 결과



6. (예제 5-11) 깊은 복사 생성자를 가진 정상적인 \_person 클래스.

#pragma warning(disable:4996)

#include <iostream>

#include <cstring>

using namespace std;

class \_person {

char\* name;

int id;

public:

\_person(int id, const char\* name); //생성자

\_person(const \_person& \_person1); //복사 생성자

~\_person();

void change\_name(const char\* name);

void show() { cout << id << ',' << name << endl; }

};

\_person::\_person(int id, const char\* name) {

this->id = id;

int len = strlen(name); //문자개수

this->name = new char[len + 1];

strcpy(this->name, name); //문자열 복사

}

\_person::\_person(const \_person& \_person1) {

this->id = \_person1.id;

int len = strlen(\_person1.name);

this->name = new char[len + 1];

strcpy(this->name, \_person1.name);

cout << "복사 생성자 실행, 원본 객체 이름" << this->name << endl;

}

\_person::~\_person() {

if (name) {

delete[] name;

}

}

void \_person::change\_name(const char\* name) {

if (strlen(name) > strlen(this->name)) { return; }

strcpy(this->name, name);

}

int main() {

\_person father(1, "kitae");

\_person daughter(father);

cout << "daughter 객체 생성 직후 ---" << endl;

father.show();

daughter.show();

daughter.change\_name("grace");

cout << "duaghter 이름을 grace로 변경한 후 ---" << endl;

father.show();

daughter.show();

return 0;

}

🡪출력 결과

