#include<iostream>

using namespace std;

//class Rectangle {

// int length;

// int breadth;

//

//public:

// Rectangle(int length,int breadth):

// length(length),breadth(breadth)

// {

//

// }

//

// friend void Calculate(Rectangle);

//};

//

//void Calculate(Rectangle r) {

// cout << "Area : " << r.breadth \* r.length << endl;

//}

//

//

//void main() {

// Rectangle r(10, 20);

// Calculate(r);

//

//

//

//}

//class Car {

//private:

// int year;

// char\* model;

// char\* vendor;

//public:

// Car(const int& year, const char\* model, const char\* vendor) {

// SetYear(year);

// SetModel(model);

// SetVendor(vendor);

// }

// void SetYear(const int& year) {

// this->year = year;

// }

// void SetModel(const char\*model) {

// int l = strlen(model);

// this->model = new char[l + 1]{};

// strcpy\_s(this->model, l + 1, model);

// }

//

// void SetVendor(const char\* v) {

// int l = strlen(v);

// this->vendor = new char[l + 1]{};

// strcpy\_s(this->vendor, l + 1, v);

// }

//

// void operator()() {

// if (model == nullptr || vendor == nullptr)return;

// cout << "+++++++++++CAR++++++++++++++" << endl;

// cout << "Model : " << model << endl;

// cout << "Vendor : " << vendor << endl;

// cout << "Year : " << year << endl;

// }

// explicit operator int() {

// return year;

// }

//

// explicit operator bool() {

// if (year >= 2010)return true;

// return false;

// }

//};

//

//void main() {

// Car c(2002, "Cruze", "Chevy");

// c();

// int d = (int)c;

// cout << d << endl;

// if (c) {

// cout << "New car" << endl;

// }

// else {

// cout << "Old car" << endl;

// }

//}

#include<assert.h>

//new

//new[]

//delete

//delete[]

class A {

int\* data;

public:

A() = default;

static int count;

A(int\* data) :data(data) {

++count;

cout << "CTOR params" << endl;

}

A(const A& a) {

this->data = a.data;

count++;

}

void Show()const {

cout << "Data : " << \*data << endl;

}

void\* operator new(size\_t size) {

cout << "New operator overloaded" << endl;

return malloc(size);

}

void\* operator new[](size\_t size) {

cout << "New [] operator overloaded" << endl;

int size\_c = size - 4;

int count = size\_c / sizeof(A);

assert(count <= 5 && "you can create max 5 objects");

return malloc(size);

}

void operator delete(void\* addressOfData) {

cout << "Delete operator overloaded" << endl;

if (addressOfData != nullptr) {

--count;

if (count == 0) {

free(addressOfData);

addressOfData = nullptr;

}

}

}

void operator delete[](void\* addressOfData) {

cout << "Delete[] operator overloaded" << endl;

if (addressOfData != nullptr) {

--count;

if (count == 0) {

free(addressOfData);

addressOfData = nullptr;

}

}

}

~A()

{

//delete data;

cout << "Destructor works" << endl;

}

};

int A::count = 0;

//void main() {

// //A\* a = new A[4]{};

// A a;

// cout << a << endl;

//

//}