|  |
| --- |
| **2019년 11월 19일 실습보고서** |
| **7조 : 송재원, 조윤직, 양석준, 박진영** |
| **실습자료1 : arraydataBack** |
| **소스코드** |
| #include <iostream>  #include "ArrayDataBak.h"  using namespace std;  ostream& operator<<(ostream& out, ArrayDataBak& data) {  data.showData();  return out;  }  int main()  {  cout << "7조 박진영 양석준 송재원 조윤직\n";    ArrayDataBak arr(5);  arr.addElement(10.2);  arr.addElement(4.2);  arr.addElement(12.2);  arr.addElement(11.2);  cout << arr;  arr.backup();  arr.emptyArray();  arr.showData();  arr.restore();  arr.showData();  cout << "========백업 확인======\n";  ArrayDataBak arr2(10);  arr2.addElement(10);  arr2.addElement(10);  arr2.addElement(10);  arr2.showData();  arr2.emptyArray();  arr2.restore();  arr2.showData();  cout << "========이동 생성자======\n";  ArrayDataBak arr3 = ArrayDataBak::getCopyInstance(arr2);  cout << "========대입 연산자======\n";  ArrayDataBak arr4;  arr4 = arr2;  cout << arr4;  cout << "========이동 대입 연산자======\n";  arr4 = ArrayDataBak::getCopyInstance(arr);  cout << arr4;  cout << "프로그램 종료\n";    }  ----------------------------------------------------------------------------  ArrayData.h  #pragma once  class ArrayData  {  private:  double\* data;  int capacity;  int used;  static int count;  public:  static int getCount();  ArrayData(const int& capacity);  ArrayData(const ArrayData& arr);  ArrayData(ArrayData&& arr);    ArrayData();  virtual ~ArrayData();  void addElement(double num);  bool full() const;  int getCapacity() const;  int& getCapacity();  int getUsed() const ;  int& getUsed() ;  void emptyArray();  void showData() const;  void operator=(const ArrayData& copy);  void operator=( ArrayData&& copy);  friend double getArraySum(const ArrayData& arr);  ArrayData getObject() {  return \*this;  }  double& operator[](const int& i);  static ArrayData getCopyInstance(const ArrayData&copy) ;  };  ----------------------------------------------------------------------------  ArrayData.cpp  #include "ArrayData.h"  #include<iostream>  using namespace std;  int ArrayData::getCount()  {  return count;  }  ArrayData::ArrayData(const int& capacity):capacity(capacity), used(0)  {  cout << this->capacity << "배열 생성\n";  count++;  this->data = new double[capacity];  }  ArrayData::ArrayData(const ArrayData& arr) : capacity(arr.capacity),used(arr.used)  {  cout << this->capacity << "배열 복사\n";  count++;  this->data = new double[this->capacity];  for (int i = 0; i < used; i++)  data[i] = arr.data[i];  }  ArrayData::ArrayData(ArrayData&& copy) :capacity(copy.capacity), used(copy.used), data(copy.data)  {  cout << this->capacity << "이동 생성자\n";  copy.data = nullptr;  }  ArrayData::ArrayData() :ArrayData(5)  {  }  int ArrayData::count = 0;  ArrayData::~ArrayData()  {    //cout << this->capacity << "배열 삭제\n";  count--;  if (data != nullptr) {  delete[] this->data;  data = nullptr;  }    }  void ArrayData::addElement(double num)  {  if (!full())  this->data[used++] = num;  else  cout << "빈 공간이 없음\n";    }  bool ArrayData::full() const  {  return capacity==used;  }  int ArrayData::getCapacity() const  {  return this->capacity;  }  int& ArrayData::getCapacity()  {  return this->capacity;  // TODO: 여기에 반환 구문을 삽입합니다.  }  int ArrayData::getUsed() const  {  return this->used;  }  int& ArrayData::getUsed()  {  // TODO: 여기에 반환 구문을 삽입합니다.  return used;  }  void ArrayData::emptyArray()  {  this->used = 0;  }  void ArrayData::showData() const  {  double\* p = this->data;  double\* endp = p + this->used;  cout << "배열 출력 :";  while (p < endp) {  cout << \*(p++)<<" ";    }  cout << endl;  }  void ArrayData::operator=(const ArrayData& copy)  {  cout << "부모 대입연산자\n";  if (capacity != copy.capacity) {  delete data;  capacity = copy.capacity;  data = new double[copy.capacity];  }  emptyArray();  for (int i = 0; i < copy.used;i++) {  addElement(\*(copy.data + i));  }  }  void ArrayData::operator=( ArrayData&& copy)  {  cout << "부모 이동 연산자 수행\n";  if (data != NULL) {  delete data;  }  capacity = copy.capacity;  used = copy.used;  data = copy.data;  copy.data = NULL;  }  double& ArrayData::operator[](const int& i)  {  return data[i];  }  ArrayData ArrayData::getCopyInstance(const ArrayData& copy)  {  ArrayData arr(copy);  return arr;  }  ----------------------------------------------------------------------------  ArrayDataBak.h  #pragma once  #include "ArrayData.h"  class ArrayDataBak :  public ArrayData  {  private:  double\* backdata;  int usedB;  public:  ArrayDataBak();  ArrayDataBak(const int& capacity);  ArrayDataBak(const ArrayDataBak& copy);  ArrayDataBak( ArrayDataBak&& copy);  ~ArrayDataBak();  void backup();  void restore();  static ArrayDataBak getCopyInstance(const ArrayDataBak& copy);  void operator=(const ArrayDataBak& copy);  void operator=(ArrayDataBak&& copy);  };  ----------------------------------------------------------------------------  ArrayDataBak.cpp  #include "ArrayDataBak.h"  #include <iostream>  using namespace std;  ArrayDataBak::ArrayDataBak() :ArrayDataBak(5)  {    }  ArrayDataBak::ArrayDataBak(const int& capacity):ArrayData(capacity)  {  cout << "백업데이터생성자" << endl;  backdata = new double[getCapacity()];  usedB = 0;  }  ArrayDataBak::ArrayDataBak(const ArrayDataBak& copy):ArrayData(copy),usedB(copy.usedB)  {  cout << this->usedB << "백업데이터 복사 생성자" << endl;  this->backdata = new double[this->usedB];  for (int i = 0; i < usedB; i++)  backdata[i] = copy.backdata[i];  }  ArrayDataBak::ArrayDataBak( ArrayDataBak&& copy) :ArrayData(copy), usedB(copy.usedB), backdata(copy.backdata)  {  cout << copy.usedB << "백업이동 생성자\n";  copy.backdata = nullptr;  }  ArrayDataBak::~ArrayDataBak()  {  //cout << "백업 데이터 삭제" << endl;  if (backdata != nullptr) {  delete[] backdata;  }  }  void ArrayDataBak::backup()  {  cout << "백업 실행\n";  usedB = getUsed();  for (int i = 0; i < usedB; i++) {  backdata[i] = (\*this)[i]; //연산자오버로딩  }  }  void ArrayDataBak::restore()  {  cout << "백업복구 실행\n";  getUsed() = usedB;  for (int i = 0; i < usedB; i++) {  (\*this)[i] = backdata[i]; //연산자오버로딩  }  }  ArrayDataBak ArrayDataBak::getCopyInstance(const ArrayDataBak& copy)  {  ArrayDataBak arr(copy);  return arr;  }  void ArrayDataBak::operator=(const ArrayDataBak& copy)  {  cout << "자식 복사대입연산자실행" << endl;  if (getCapacity() != copy.getCapacity()) {  delete [] backdata;  getCapacity() = copy.getCapacity();  backdata = new double[copy.getCapacity()];  }  usedB = copy.usedB;  for (int i = 0; i < copy.usedB; i++) {  backdata[i] = copy.backdata[i];  }  ArrayData::operator=(copy);  }  void ArrayDataBak::operator=(ArrayDataBak&& copy)  {  cout << "백업 이동 연산자 수행\n";  ArrayData::operator=(copy);  if (backdata != NULL) {  delete[] backdata;  }  usedB = copy.usedB;  backdata = copy.backdata;  copy.backdata = NULL;  } |
| **실행결과** |
|  |
| 이동 생성자시에 부모의 이동생성자가 아닌 복사생성자가 실행되는 이유는 찾지 못함 |