|  |
| --- |
| **2019년 11월 28일 실습보고서** |
| **7조 조윤직 송재원 양석준 진영** |
| **실습자료1 : [템플릿상속,STL벡터]** |
| **소스코드** |
| //TArrayDataBak.h  #pragma once  #include "ArrayData.h"  template <typename T>  class TArrayDataBak :  public TArrayData<T>  {  private:  T\* backdata;  int usedB;  public:  TArrayDataBak();  TArrayDataBak(const int& capacity);  TArrayDataBak(const TArrayDataBak& copy);  TArrayDataBak(TArrayDataBak&& copy);  ~TArrayDataBak();  void backup();  void restore();  void operator=(const TArrayDataBak& copy);  void operator=(TArrayDataBak&& copy);  T& operator[] (const int& n);  };  template<typename T>  TArrayDataBak<T>::TArrayDataBak():TArrayData()  {  usedB = 0;  backdata = new T[this->getCapacity()];  }  template<typename T>  TArrayDataBak<T>::TArrayDataBak(const int& capacity): TArrayData<T>(capacity)  {//부모 생성자에 <T>를 적어야 오류가 안나 더라..  usedB = 0;  backdata = new T[capacity];  }  template<typename T>  inline TArrayDataBak<T>::TArrayDataBak(const TArrayDataBak& copy):TArrayData<T>(copy)  {  usedB = copy.usedB;  backdata = new T[this->capacity];  for (int i = 0; i < usedB; i++) {  backdata[i] = copy.backdata[i];  }  }  template<typename T>  inline TArrayDataBak<T>::TArrayDataBak(TArrayDataBak&& copy):TArrayData<T>(move(copy))  {  usedB = copy.usedB;  backdata = copy.backdata;  copy.backdata = NULL;  }  template<typename T>  inline TArrayDataBak<T>::~TArrayDataBak()  {  if (backdata != NULL) {  delete[] backdata;  }  }  template<typename T>  inline void TArrayDataBak<T>::backup()  {  usedB = this->getUsed();  for (int i = 0; i < usedB; i++) {  backdata[i] = this->data[i];  }  }  template<typename T>  inline void TArrayDataBak<T>::restore()  {  this->used = usedB;  for (int i = 0; i < this->used; i++) {  this->data[i] = backdata[i];  }  }  template<typename T>  inline void TArrayDataBak<T>::operator=(const TArrayDataBak& copy)  {  if (backdata != NULL) {  delete[] backdata;  }  usedB = copy.usedB;  backdata = new T[copy.capacity];  for (int i = 0; i < usedB; i++) {  backdata[i] = copy.backdata[i];  }  }  template<typename T>  inline void TArrayDataBak<T>::operator=(TArrayDataBak&& copy)  {  if (backdata != NULL)  delete[] backdata;  backdata = new T[copy.capacity];  usedB = copy.usedB;  TArrayData<T>::operator=(copy);  for (int i = 0; i < usedB; i++) {  backdata[i] = copy.backdata[i];  }  }  template<typename T>  inline T& TArrayDataBak<T>::operator[](const int& n)  {  return data[n];  }  // ArrayData.h  #pragma once  #include <iostream>  using namespace std;  template<typename T>  class TArrayData  {  private:  T\* data;  int capacity;  int used;  public:  TArrayData(const int& capacity);  TArrayData(const TArrayData& arr);  TArrayData(TArrayData&& arr);  TArrayData();  ~TArrayData();  void addElement(T num);  bool full() const;  int getCapacity() const;  int getUsed() const {  return used;  }  void emptyArray();  void showData() const;  void operator=(const TArrayData& copy);  void operator=(TArrayData&& copy);  TArrayData getObject() {  return \*this;  }    };  template<typename T>  inline TArrayData<T>::TArrayData(const int& capacity)  {  used = 0;  this->capacity = capacity;  data = new T[capacity];  }  template<typename T>  inline TArrayData<T>::TArrayData(const TArrayData& arr):capacity(arr.capacity),used(arr.used)  {  data = new T[capacity];  for (int i = 0; i < used; i++) {  data[i] = arr.data[i];  }  }  template<typename T>  inline TArrayData<T>::TArrayData(TArrayData&& arr):used(arr.used),capacity(arr.capacity)  {  data = arr.data;  arr.data = NULL;  }  template<typename T>  inline TArrayData<T>::TArrayData() :TArrayData(10)  {  }  template<typename T>  inline TArrayData<T>::~TArrayData()  {  if (data != NULL) {  delete[] data;  }  }  template<typename T>  inline void TArrayData<T>::addElement(T num)  {  if (!full())  data[used++] = num;  else  cout << "data 꽉 참\n";  }  template<typename T>  inline bool TArrayData<T>::full() const  {  return used >= capacity;  }  template<typename T>  inline int TArrayData<T>::getCapacity() const  {  return capacity;  }  template<typename T>  inline void TArrayData<T>::emptyArray()  {  used = 0;  }  template<typename T>  inline void TArrayData<T>::showData() const  {  cout << "used :" << used << "배열 출력\n";  for (int i = 0; i < used; i++) {  cout << i<<" : "<<data[i] << endl;  }  }  template<typename T>  inline void TArrayData<T>::operator=(const TArrayData& copy)  {  if (data != NULL) {  delete[] data;  }  data = new T[copy.capacity];  capacity = copy.capacity;  used = copy.used;  for (int i = 0; i < used; i++) {  data[i] = copy.data[i];  }  }  template<typename T>  inline void TArrayData<T>::operator=(TArrayData&& copy)  {  if (data != NULL) {  delete[] data;  }    capacity = copy.capacity;  used = copy.used;  data = copy.data;  copy.data = NULL;  }  //student.h  #pragma once  #include<iostream>  using namespace std;  class Student  {  private:  string name;  int score;  public:  Student() :Student("", 0) {  }  Student(string name, int score) :name(name), score(score) {  }  Student(string name) :Student(name, 0) {  }  friend ostream& operator<<(ostream& out, Student& copy);    bool operator==(string check){  return this->name == check;  }  bool operator==(Student& check) {  return (this->name == check.name && this->score == check.score);  }  };  //main.cpp  #include "Student.h"  #include "ArrayData.h"  #include "TArrayDataBak.h"  #include <vector>  using namespace std;  ostream& operator<<(ostream& out, Student& copy) {  out << "이름 : " << copy.name << ", 성적 : " << copy.score;  return out;  }  //studen\* 출력 함수  ostream& operator<<(ostream& out,const Student\* copy) {  out << "이름 : " << copy->name << ", 성적 : " << copy->score;  return out;  }  template<typename T3>  ostream& operator<<(ostream& out, const TArrayData<T3>& copy) {  copy.showData();  return out;  }  template<typename T, typename T2>  bool Search(T\* arr, int sizearr, T2 find, int& index) {  for (int i = 0; i < sizearr; i++) {  if (arr[i] == find) {  index = i;  return true;  }  }  return false;  cout << "값 찾지 못함\n";  }  template<typename T4>  void printArr(const vector<T4>& v) {  //vector를 const로 받아와서 -- const\_iterator로 안하면 오류 발생  typename vector<T4>::const\_iterator itt;  for (itt = v.begin(); itt != v.end(); itt++) {  cout << \*itt << endl;  }  }  //배열 동적할당 다 삭제해주는 함수  template<typename T1>  void deleteVectorele(vector<T1>& v) {  //v가 비워질 때까지 실행  while (!v.empty())  {  //동적할당 제거, vector 원소-팝  delete v.back();  v.pop\_back();  }  }  int main()  {  /\*  //실습 1  TArrayDataBak<Student> std(10);  std.addElement(Student("greenjoa1", 50));  std.addElement(Student("greenjoa2", 20));  std.addElement(Student("greenjoa3", 30));  std.showData();  cout << "백업\n";  std.backup();  cout << "원소 추가\n";  std.addElement(Student("greeen", 10));  std.showData();  cout << "====복구====\n";  std.restore();  std.showData();  \*/    //실습2 -vector 출력 템플릿함수-iterator 사용  //vector<int > intarr;  //intarr.push\_back(10);  //intarr.push\_back(10);  //intarr.push\_back(10);  //intarr.push\_back(10);  //intarr.push\_back(10);  //intarr.push\_back(10);  //printArr(intarr);  //vector<double> doublearr;  //doublearr.push\_back(10.1);  //doublearr.push\_back(10.1);  //doublearr.push\_back(10.1);  //doublearr.push\_back(10.1);  //printArr(doublearr);  //실습 3 -stdarr 동적할당 해제, 원소 삭제함수  //vector<Student\*> stdarr;  //stdarr.push\_back(new Student("greanjoa1", 20));  //stdarr.push\_back(new Student("greanjoa2", 20));  //stdarr.push\_back(new Student("greanjoa3", 20));  //stdarr.push\_back(new Student("greanjoa4", 20));  //stdarr.push\_back(new Student("greanjoa5", 20));  //printArr(stdarr);  ////  //deleteVectorele(stdarr);  //  //printArr(stdarr);  } |
| **실행결과** |
|  |
|  |