|  |  |  |
| --- | --- | --- |
| American University of SharjahSchool of Engineering Computer Engineering Department  P. O. Box 26666 Sharjah, UAE |  | Lab Instructor: Eng. Sameer Alawnah  **Office**: EB2 - 101  **Phone**: 06-515 2974  **e-mail**: salawnah@aus.edu  **Semester**: Fall 2016 |

**CMP 220L – Introduction to Computer Science II**

**Inheritance with pointers**

**LAB Assignment 11**

You have the following class headers and a driver:

//This is the header file pfarray2d.h. This is the interface for the class

//PFArray2D. Objects of this type are partially filled arrays of doubles.

#ifndef PFARRAY2D\_H

#define PFARRAY2D\_H

class PFArray2D

{

public:

PFArray2D( );

//Initializes with a capacity of 50.

PFArray2D(int capacityValue);

PFArray2D(const PFArray2D& pfaObject);

void addElement(double element,int row);

//add element to the array[row]

bool full( int row) const;

//Returns true if the array[row] is full, false otherwise.

int getCapacity( ) const;

int getNumberUsed( int row) const;

// get the number of items in array[row]

void emptyArray( );

//Resets the number used=0, effectively emptying the array.

double at(int row,int col);

//return the element at array[row][col]

PFArray2D& operator =(const PFArray2D& rightSide);

~PFArray2D( );

protected:

double \*a[3]; //for a 2D array of doubles.

int capacity; //for the number of items in every row of the array.

int used[3]; //for the number of array positions currently in use in every row.

};

#endif //PFARRAY2D\_H

//This is the header file pfarray2dbak.h. This is the interface for the class

//PFArray2DBak. Objects of this type are partially filled arrays of doubles.

//This version allows the programmer to make a backup copy and restore

//to the last saved copy of the partially filled array.

#ifndef PFARRAY2DBAK\_H

#define PFARRAY2DBAK\_H

#include "pfarray2d.h"

class PFArray2DBak : public PFArray2D

{

public:

PFArray2DBak( );

//Initializes with a capacity of 50.

PFArray2DBak(int capacityValue);

PFArray2DBak(const PFArray2DBak& Object);

void backup( );

//Makes a backup copy of the partially filled 2D array.

void restore( );

//Restores the partially filled array to the last saved version.

//If backup has never been invoked, this empties the partially filled array.

PFArray2DBak& operator =(const PFArray2DBak& rightSide);

~PFArray2DBak( );

private:

double \*b[3]; //for a backup of main array.

int usedB[3]; //backup for inherited member variable used.

};

#endif //PFARRAY2DBAK\_H

//Program to demonstrate the class PFArray2DBak.

#include <iostream>

#include "pfarray2dbak.h"

using std::cin;

using std::cout;

using std::endl;

void testPFArray2DBak( );

//Conducts one test of the class PFArray2DBak.

int main( )

{

cout << "This program tests the class PFArray2DBak.\n";

char ans;

do

{

testPFArray2DBak( );

cout << "Test again? (y/n) ";4

cin >> ans;

}while ((ans == 'y') || (ans == 'Y'));

return 0;

}

void testPFArray2DBak( )

{

int cap;

cout << "Enter capacity of this super array: ";

cin >> cap;

PFArray2DBak backupArr(cap);

for(int row=0;row<3;row++)

{

cout << "Enter up to " << cap << " nonnegative numbers into ROW "<<row<<".\n";

cout << "Place a negative number at the end.\n";

double next;

cin >> next;

while ((next >= 0) && (!backupArr.full(row)))

{

backupArr.addElement(next,row);

cin >> next;

}

if (next >= 0)

{

cout << "Could not read all numbers.\n";

//Clear the unread input:

while (next >= 0)

cin >> next;

}

}

cout<<"You have the following numbers read and stored:\n";

for(int row=0;row<3;row++)

{

cout<<"ROW "<<row<<" : "<<backupArr.getNumberUsed(row)<<" number \n";

}

cout<<"Array Content :\n";

for(int row=0;row<3;row++)

{

cout<<"ROW "<<row<<" : ";

for(int col=0;col<backupArr.getNumberUsed(row);col++)

{

cout<<backupArr.at(row, col)<<" ";

}

cout<<"\n";

}

cout << "Backing up array.\n";

backupArr.backup();

cout << "emptying array.\n";

backupArr.emptyArray();

cout<<"You have the following numbers read and stored:\n";

for(int row=0;row<3;row++)

{

cout<<"ROW "<<row<<" : "<<backupArr.getNumberUsed(row)<<" number \n";

}

cout << "Restoring array.\n";

backupArr.restore();

cout<<"You have the following numbers read and stored:\n";

for(int row=0;row<3;row++)

{

cout<<"ROW "<<row<<" : "<<backupArr.getNumberUsed(row)<<" number \n";

}

cout<<"Array Content :\n";

for(int row=0;row<3;row++)

{

cout<<"ROW "<<row<<" : ";

for(int col=0;col<backupArr.getNumberUsed(row);col++)

{

cout<<backupArr.at(row, col)<<" ";

}

cout<<"\n";

}

}

Implement all unimplemented functions and operators in both classes.

Execute the main and test it with some inputs, a sample run is given below:

Note: You can make use of the book’s example available on ilearn.

**This program tests the class PFArray2DBak.**

**Enter capacity of this super array:** 4

**Enter up to 4 nonnegative numbers into ROW 0.**

**Place a negative number at the end.**

1 2 -1

**Enter up to 4 nonnegative numbers into ROW 1.**

**Place a negative number at the end.**

4 5 3 -1

**Enter up to 4 nonnegative numbers into ROW 2.**

**Place a negative number at the end.**

2 3 4 9 3

**Could not read all numbers.**

3

-1

**You have the following numbers read and stored:**

**ROW 0 : 2 number**

**ROW 1 : 3 number**

**ROW 2 : 4 number**

**Array Content :**

**ROW 0 : 1 2**

**ROW 1 : 4 5 3**

**ROW 2 : 2 3 4 9**

**Backing up array.**

**emptying array.**

**You have the following numbers read and stored:**

**ROW 0 : 0 number**

**ROW 1 : 0 number**

**ROW 2 : 0 number**

**Restoring array.**

**You have the following numbers read and stored:**

**ROW 0 : 2 number**

**ROW 1 : 3 number**

**ROW 2 : 4 number**

**Array Content :**

**ROW 0 : 1 2**

**ROW 1 : 4 5 3**

**ROW 2 : 2 3 4 9**

**Test again? (y/n)**

Good Luck ☺

#include"PFArray2D.h"

#include<iostream>

using namespace std;

PFArray2D::PFArray2D()//Initializes with a capacity of 50.

{

capacity = 50;

for (int i = 0; i < 3; i++)

{

a[i] = new double[capacity];

used[i] = 0;

}

}

PFArray2D::PFArray2D(int capacityValue)

{

for (int i = 0; i < 3; i++)

{

a[i] = new double[capacityValue];

used[i] = 0;

}

}

PFArray2D::PFArray2D(const PFArray2D& pfaObject)

{

capacity = pfaObject.getCapacity();

for (int i = 0; i < 3; i++)

{

used[i] = pfaObject.used[i];

a[i] = new double[capacity];

for (int j = 0; j < used[i]; j++)

{

a[i][j] = pfaObject.a[i][j];

}

}

}

void PFArray2D::addElement(double element, int row)

{

if (used[row] >= capacity)

{

cout << "capacity exceeded" << endl;

exit(1);

}

a[row][used[row]] = element;

used[row]++;

}

//add element to the array[row]

bool PFArray2D::full(int row) const

{

return(capacity == used[row]);

}

//Returns true if the array[row] is full, false otherwise.

int PFArray2D::getCapacity() const

{

return(capacity);

}

int PFArray2D::getNumberUsed(int row) const

{

return (used[row]);

}

// get the number of items in array[row]

void PFArray2D::emptyArray()

{

for (int i = 0; i < 3; i++)

{

used[i] = 0;

}

}

//Resets the number used=0, effectively emptying the array.

double PFArray2D::at(int row, int col)

{

return(a[row][col]);

}

//return the element at array[row][col]

PFArray2D& PFArray2D::operator =(const PFArray2D& rightSide)

{

if (capacity != rightSide.capacity)

{

for (int i = 0; i < 3; i++)

{

delete[] a[i];

a[i] = new double[rightSide.capacity];

}

}

for (int i = 0; i < 3; i++)

{

capacity = rightSide.capacity;

used[i] = rightSide.used[i];

for (int j = 0; j < used[i]; j++)

{

a[i][j] = rightSide.a[i][j];

}

}

return \*this;

}

PFArray2D::~PFArray2D()

{

for (int i = 0; i < 3; i++)

{

delete[] a[i];

}

}

#include"PFArray2DBak.h"

#include<iostream>

using namespace std;

PFArray2DBak::PFArray2DBak()

{

capacity = 50;

for (int i = 0; i < 3; i++)

{

usedB[i] = 0;

b[i] = new double[capacity];

}

}

//Initializes with a capacity of 50.

PFArray2DBak::PFArray2DBak(int capacityValue)

{

capacity = capacityValue;

for (int i = 0; i < 3; i++)

{

usedB[i] = 0;

b[i] = new double[capacity];

}

}

PFArray2DBak::PFArray2DBak(const PFArray2DBak& Object)

{

capacity = Object.getCapacity();

for (int i = 0; i < 3; i++)

{

usedB[i] = Object.usedB[i];

b[i] = new double[capacity];

for (int j = 0; j < usedB[i]; j++)

{

b[i][j] = Object.b[i][j];

}

}

}

void PFArray2DBak::backup()

{

for (int i = 0; i < 3; i++)

{

usedB[i] = used[i];

for (int j = 0; j < usedB[i]; j++)

{

b[i][j] = a[i][j];

}

}

}

//Makes a backup copy of the partially filled 2D array.

void PFArray2DBak::restore()

{

emptyArray();

for (int i = 0; i < 3; i++)

{

used[i] = usedB[i];

for (int j = 0; j < used[i]; j++)

{

a[i][j] = b[i][j];

}

}

}

//Restores the partially filled array to the last saved version.

//If backup has never been invoked, this empties the partially filled array.

PFArray2DBak& PFArray2DBak::operator = (const PFArray2DBak& rightSide)

{

if (capacity != rightSide.capacity)

{

for (int i = 0; i < 3; i++)

{

delete[] b[i];

b[i] = new double[rightSide.getCapacity()];

}

}

capacity = rightSide.capacity;

for (int i = 0; i < 3; i++)

{

usedB[i] = rightSide.usedB[i];

for (int j = 0; j < usedB[i]; j++)

{

b[i][j] = rightSide.b[i][j];

}

}

return \*this;

}

PFArray2DBak::~PFArray2DBak()

{

for (int i = 0; i < 3; i++)

{

delete[] b[i];

}

}

