חורף 2012 מועד ב פתרון

שאלה 1.

Solution:

#ifndef \_ARRAY\_H\_

#define \_ARRAY\_H\_

#include <stdlib.h>

typedef struct Array\* PArray;

typedef enum Result {FAIL = 0, SUCCESS};

typedef void\* PElement;

PArray CreateArray(unsigned int N);

void DestroyArray(PArray);

Result ArraySetAt(PArray, unsigned int i, PElement pElem);

PElement ArrayGetAt(PArray, unsigned int i);

struct Array

{

PElement\* \_array;

unsigned int \_N;

};

PArray CreateArray(unsigned int N)

{

unsigned int i;

PArray pArr = (PArray)malloc(sizeof(struct Array));

pArr->\_array = (PElement\*)malloc(sizeof(PElement)\*N);

for (i=1; i<=N; i++)

pArr->\_array[i] = NULL;

pArr->\_N = N;

return pArr;

}

Result ArraySetAt(PArray pArr, unsigned int i, PElement pElem)

{

if (i > pArr->\_N)

return FAIL;

pArr->\_array[i]=&pElem;

return SUCCESS;

}

#endif

#ifndef \_MATRIX\_H\_

#define \_MATRIX\_H\_

#include "array.h"

typedef struct Matrix\* PMatrix;

typedef void\* PElement;

typedef PElement (\*MultFunc)(PElement pl, PElement pr);

typedef PElement (\*SumFunc)(PElement pl, PElement pr);

PMatrix CreateMatrix(unsigned int N, unsigned int M);

void DestroyMatrix(PMatrix);

void MatrixSetAt(PMatrix, unsigned int i, unsigned int j, PElement pElem);

PElement MatrixGetAt(PMatrix, unsigned int i, unsigned int j);

struct Matrix

{

PArray \_pArr;

unsigned int \_M;

unsigned int \_N;

};

PMatrix CreateMatrix(unsigned int N, unsigned int M)

{

unsigned int i;

PMatrix pMat = (PMatrix)malloc(sizeof(struct Matrix));

pMat->\_pArr = CreateArray(N);

PArray pArr;

for (i=1; i<=N; i++) {

pArr = CreateArray(M);

ArraySetAt(pMat->\_pArr, i, &pArr);

}

pMat->\_N = N;

pMat->\_M = M;

return pMat;

}

PElement MatrixGetAt(PMatrix pm, unsigned int i, unsigned int j)

{

return ArrayGetAt(\*(PArray\*)ArrayGetAt(pm->\_pArr, i), j);

}

#endif

שאלה 2.

#include <iostream>

#include <exception>

using namespace std;

template<class T>

class Multival{

public:

// solution for part 1: define these functions

Multival(const T& \_val);

Multival(const Multival<T>& rhs);

~Multival();

Multival& operator=(const Multival& rhs);

Multival& operator+=(const Multival& rhs);

Multival& operator|=(const Multival& rhs);

Multival& operator|=(const T& rhs);

Multival& operator+=(const T& rhs);

// print is for debugging, not a part of the exrcize:

ostream& print(ostream& ost) const{

ost << vals[0];

for(int i=1;i<numVals;++i)

ost << " or " << vals[i];

return ost;

}

private:

T\* vals;

int numVals;

};

//solution for part 2:

template<class T>

Multival<T>& Multival<T>::operator+=(const Multival<T>& rhs) {

int newNumVals=numVals\*rhs.numVals;

T\* newVals=new T[newNumVals];

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

for(int j=0;j<rhs.numVals;j++)

newVals[i+j\*numVals]=vals[i]+rhs.vals[j];

delete [] vals;

vals=newVals;

numVals=newNumVals;

return \*this;

}

//solution for part 3

template<class T>

Multival<T>& Multival<T>::operator|=(const T& rhs) {

try {

int newNumVals=numVals+1;

T\* newVals=new T[newNumVals];

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

newVals[i]=vals[i];

newVals[newNumVals-1]=rhs;

delete [] vals;

vals=newVals;

numVals=newNumVals;

} catch (bad\_alloc) {

delete[] vals;

vals=new T[1];

vals[0]=rhs;

numVals=1;

}

return \*this;

}

//other functions - no implementation required from students

template<class T>

Multival<T>::Multival(const T& \_val): vals(new T[1]), numVals(1) {vals[0]=\_val;}

template<class T>

Multival<T>::Multival(const Multival<T>& rhs): vals(new T[rhs.numVals]), numVals(rhs.numVals) {

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

vals[i]=rhs.vals[i];

}

template<class T>

Multival<T>::~Multival() {

delete [] vals;

}

template<class T>

Multival<T>& Multival<T>::operator=(const Multival<T>& rhs) {

if (this != &rhs) {

delete [] vals;

numVals=rhs.numVals;

vals=new T[numVals];

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

vals[i]=rhs.vals[i];

}

return \*this;

}

template<class T>

Multival<T>& Multival<T>::operator|=(const Multival<T>& rhs) {

int newNumVals=numVals+rhs.numVals;

T\* newVals=new T[newNumVals];

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

newVals[i]=vals[i];

for(int i=0;i<rhs.numVals;++i)

newVals[i+numVals]=rhs.vals[i];

delete [] vals;

vals=newVals;

numVals=newNumVals;

return \*this;

}

template<class T>

Multival<T>& Multival<T>::operator+=(const T& rhs) {

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

vals[i]+=rhs;

return \*this;

}

template<class T>

ostream& operator<<(ostream& lhs, const Multival<T>& rhs){

return rhs.print(lhs);

}

int main() {

Multival<int> a(1), b(10); //a is 1, b is 10, c is 0

a|=2; //a is 1 or 2

b|=20; //b is 10 or 20

Multival<int> c(b); //c is 10 or 20

c+=20; //c is 30 or 40

b|=c; //b is 10 or 20 or 30 or 40

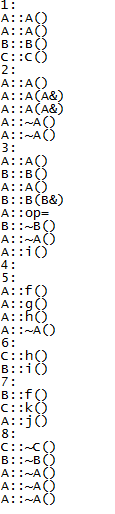
a+=b; //a is 11 or 12 or 21 or 22 or 31 or 32 or 41 or 42

cout << "a is " << a << endl;

return 0;

}

שאלה 3 **פתרון (לפי הרצה ב- Unix):**



שאלה 4

חלק א': BASH (10 נק')

#!/bin/bash

lines=0

if [[ $1 == "-all" ]]; then

for file in \*; do

if [[ -f $file ]]; then

tot=`wc -l ${file}`

stot=( $tot )

(( lines=${lines} + ${stot[0]} ))

fi

done

else

for file in $\*; do

if [[ -f ${file} ]]; then

tot=`wc -l ${file}`

stot=( $tot )

(( lines=${lines} + ${stot[0]} ))

fi

done

fi

echo $lines

פתרון חלק ב'

1. חפיפת operator= מציבה null למצביע שהעתקנו ממנו, וכך לא נגרם מצב של 2 מצביעים לאותו זיכרון דינאמי, ושחרור הזיכרון של המצביע שהעתקנו ממנו לא תגרום למצביע החדש להיות dangling.
2. הפרמטר הראשון בכל פונקציה של ה-ADT הינו מצביע לאובייקט.
3. במקום דגל (בינארי) נשתמש במונה, אשר יספור עד ל-5 אובייקטים ואז יפסיק לייצר ישויות חדשות, ימשיך לעלות ב-1 עבור כל בקשה לאובייקט חדש וירד ב-1 בכל פעם שמשתחרר אובייקט. כאשר נגיע ל-5 אובייקטים נתחיל לשחרר בכל פעם שהמונה יורד ב-1.
4. לסובייקט אין שליטה בכמות המאזינים לו, שיכולה לגדול ולגדול. אם יהיו הרבה מאזינים ובכל שינוי תשלחנה הודעות עדכון לכולם, יתכן עומס רב ופגיעה בביצועים.