**חורף-2014-מועד-א--פתרון**

**שאלת ADT - 30 נק'**

סעיף א'

typedef pElement (\*CLONE\_FUNC) (pElement);

typedef pElement (\*SUM\_FUNC) (pElement, pElement);

typedef void (\*DEL\_FUNC) (pElement);

typedef void (\*PRINT\_FUNC) (pElement);

typedef struct \_multiList {

pNode \_head;

int \_num\_slices;

CLONE\_FUNC cf;

SUM\_FUNC sf;

DEL\_FUNC df;

PRINT\_FUNC pf;

} MultiList;

סעיף ב'

pMultiList CreateMultiList(CLONE\_FUNC cf, SUM\_FUNC sf, DEL\_FUNC df, PRINT\_FUNC pf);

סעיף ג'

pElement SumElements(pMultiList al, int start\_index, int end\_index) {

pElement temp;

pElement res = NULL;

pElement del;

int i;

if (!al || start\_index<0 || end\_index < start\_index)

return NULL;

for (i = start\_index; i < end\_index; i++) {

temp = GetIndex(al, i);

if (temp == NULL)

continue;

if (res == NULL)

res = al->cf(temp);

else {

del = res;

res = al->sf(res, temp);

al->df(del);

}

}

return res;

}

סעיף ד'

pIncome alloc\_income(double income, double expense) {

pIncome p = (pIncome) malloc(sizeof(Income));

p->income = income;

p->expense = expense;

return p;

}

pElement sum\_incomes (pElement a, pElement b) {

pIncome i = (pIncome)a;

pIncome j = (pIncome)b;

return (pElement) alloc\_income(i->income+j->income,i->expense + j->expense);

}

pElement clone\_income (pElement a) {

pIncome i = (pIncome)a;

return (pElement) alloc\_income(i->income, i->expense);

}

void del\_income (pElement p) {

free(p);

}

void print\_income (pElement p) {

pIncome pi = (pIncome) p;

printf("Income = %6.2f, Expense = %6.2f", pi->income, pi->expense);

}

void main() {

pMultiList al = CreateMultiList(clone\_income, sum\_incomes, del\_income,

print\_income);

}

**שאלה 2-C++**

1. תשובה: מחלקת animal, מחלקה עבור כל חיה. סה"כ 5 מחלקות.
2. תשובה: עבור num\_of\_animals צריך משתנה static במחלקה Animal, שיקודם בכל c-tor.  
   עבור roar צריך פונק' virtual במחלקה animal שתידרס(override) בכל מחלקה יורשת.
3. תשובה: שם ומשקל משותפים לכל המחלקות, לכן יהיו בAnimal. שאר המאפיינים יהיו בתוך המחלקות הספציפיות.
4. תשובה: המחלקה ninjaTurtle יורשת מ turtle. צריך לכתוב מחדש את הפונקציה roar ואת ה c-tor, ולהוסיף פרמטר weapon.
5. יש לממש את האופרטור+ כפונקציה חברה במחלקה turtle, שמקבלת animal ומחזירה פוינטר לturtle לאחר שיצרה אותו ע"י הקצאה דינאמית. הפונקציה צריכה לקבל את animal כ:   const &, אחרת יצטרכו לממש גם copy c-tor כדי לאפשר העברה by value וזה נוגד את הוראות השאלה שאומרים לממש פונקציה אחת בלבד.
6. יש לכתוב פונקציה חיצונית למחלקות: operator<<. הפונקציה תקבל מצביע ל animal, ותקרא לפונקציה print שהיא וירטואלית ושייכת למחלקה animal. כל אחד מהבנים יממש את הפונקציה print לפי ההגדרות שבפלט. יש לתת הרשאת friend מתוך animal לאופרטור << שכן הפונקציה print לא תהיה חשופה(תהיה protected).
7. לכתוב פונקציה שמשתמשת בSTL::sort, יש לכתוב function object שמשווה 2 חיות לפי השם שלהן, ולשלוח אובייקט כזה לפונקציה sort. UML: 4 החיות יורשות ממחלקה אבסטרקטית – Animal. המחלקה ninja\_turtle יורשת מ Turtle

**Zoo.h**

#include <iostream>

#include <string>

using namespace std;

class Animal

{

public:

static int num\_of\_animals; // seif 2

virtual void roar() const = 0;// seif 2

string m\_name;

friend ostream& operator<<(ostream& os, const Animal\* pAnimal);

protected:

virtual void print() const = 0; // seif 5

Animal(const string name)

{

m\_name = name;

num\_of\_animals++; // seif 2

}

int m\_weight;

};

class Lion: public Animal

{

public:

Lion(const string name):Animal(name)

{ }

int hair\_size;

virtual void roar() const

{

printf("arrrrrr \n");

}

protected:

virtual void print() const // seif 5

{

cout << "type: Lion , name: " << m\_name <<endl;

}

};

class Turtle: public Animal

{

public:

Turtle(const string name):Animal(name)

{ }

int carapace\_size;

virtual void roar() const

{

printf("Squik \n");

}

Turtle\* operator+(const Animal& animal) // seif 5

{

string new\_name = "son\_of\_" + this->m\_name + "\_and\_" + animal.m\_name;

Turtle\* new\_turtle = new Turtle(new\_name);

new\_turtle->m\_weight = 3;

new\_turtle->carapace\_size = this->carapace\_size;

return new\_turtle;

}

protected:

virtual void print() const // seif 6

{

cout << "type: Turtle , name: " << m\_name <<endl;

}

};

class Goose: public Animal

{

public:

Goose(string name):Animal(name)

{}

string origin\_country;

virtual void roar() const

{

printf("Quaq \n");

}

protected:

virtual void print() const // seif 6

{

cout << "type: Goose , name: " << m\_name <<endl;

}

};

class Bunny: public Animal

{

public:

Bunny(string name):Animal(name)

{}

int ear\_size;

virtual void roar() const

{

printf("what's up dock!? \n");

}

protected:

virtual void print() const // seif 6

{

cout << "type: Bunny , name: " << m\_name <<endl;

}

};

// seif 4:

class ninja\_turtle: public Turtle

{

public:

ninja\_turtle(const string name, int weapon):Turtle(name),ninja\_weapon(weapon)

{}

int ninja\_weapon;

virtual void roar() const

{

printf("kababanga \n");

}

protected:

virtual void print() const // seif 6

{

cout << "type: ninja\_turtle , name: " << m\_name << " , weapon: " ;

switch (ninja\_weapon)

{

case 1:

cout << "sword" << endl;

break;

case 2:

cout << "stick" << endl;

break;

case 3:

cout << "numchaku" << endl;

break;

case 4:

cout << "daggers" << endl;

break;

}

}

};

3. תשובה:

1:

A::A()

B::B()

A::A()

C::C()

2:

A::A()

B::B(B&)

A::A(A&)

C::op=

A::A(A&)

3:

A::A()

A::A()

D::D()

A::A()

A::A(A&)

D::D(D&)

4:

A::A()

B::B(B&)

B::print()

B::~B()

A::~A()

C::print()

5:

A::print()

A::A(A&)

A::print()

A::~A()

6:

D::~D()

A::~A()

A::~A()

D::~D()

A::~A()

A::~A()

7:

C::~C()

A::~A()

B::~B()

A::~A()

8:

A::~A()

C::~C()

A::~A()

B::~B()

A::~A()

4.

א

add-num

#!/bin/bash

cat tasks |./add1

add1

#! /bin/bash

num=1

while read line; do

echo $num $line

((num++))

Done

ב.

add-task

#!/bin/bash

cat tasks |./add2 $\*

add2

#! /bin/bash

while read line; do

echo $line

done

echo $\*

ג.

Delete

#!/bin/bash

cat tasks |./delete

delete

#! /bin/bash

num=1

while read line; do

if (( $num != $1)); then

echo $line

fi

((num++))

Done

הבנה:

1. פוינטר ל-ADT הספציפי שעובדים אתו. (ניתן כפרמטר ראשון בכל פונקציות הממשק)
2. לחריגה הסטנדרטית יש מתודה וירטואלית בשם what() המחזירה מחרוזת.

החריגה הנורשת דורסת את המתודה ומחזירה מחרוזת שמתאימה לתיאור החריגה הנורשת.

1. כאשר יוצאים מבלוק, המצביע החכם מושמד כמו כל משתנה פנימי ואז הדיסטרקטור שלו דואג לשחרור הזיכרון עליו הוא מצביע
2. 1. הם ממומשים כטמפלייט, ולכן מותאמים לכל טיפוס T.

2. הקונטיינרים משתפים ממשק, ולכן האלגוריתמים יכולים לעבוד על קונטיינרים שונים באותה צורה.

3. הם פועלים דרך האיטרטורים ולכן אינם תלויים במימוש הקונטיינר הספציפי.