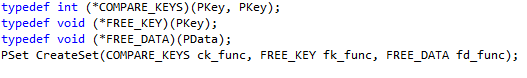
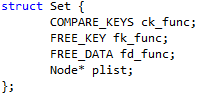
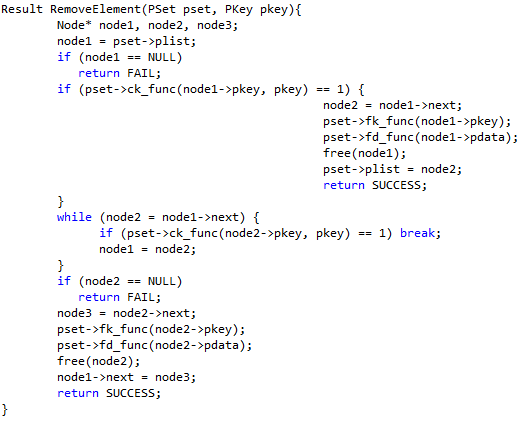
# חורף 2012 מועד ב - פתרון

פתרון שאלה 1

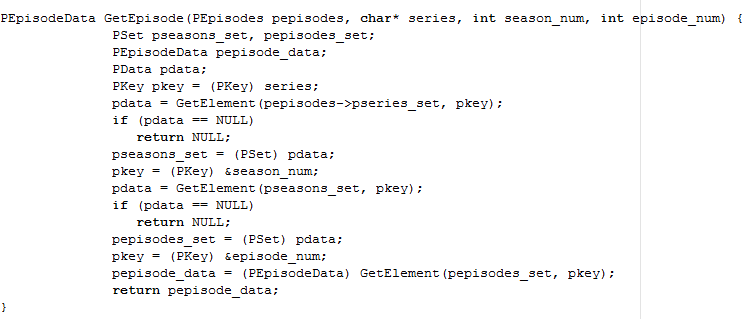


****

1. **הערה: המימוש המתקבל כאשר מניחים כי הצומת הראשון הוא צומת דמה פשוט יותר.**

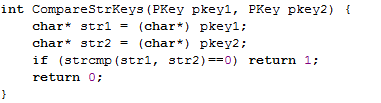
****



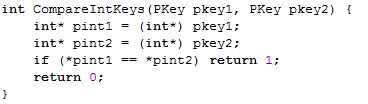


1. נדרש לממש רק פונקציות השוואת מפתחות.

עבור סט הסדרות המפתח הוא שם הסדרה (מחרוזת) ולכן פונקציית השוואת המפתחות תהיה:



עבור שאר הסטים, המפתח הוא מספר העונה או מספר הפרק (int), ולכן פונקציית השוואת המפתחות תהיה:



# שאלה 2 – C++ (30 נק')

פתרון:

#ifndef \_ARRAY\_H\_

#define \_ARRAY\_H\_

class BadIndex {};

template <unsigned int Size, class Type>

class Array

{

public:

Type val(unsigned int i)

{

if (i >= Size)

throw BadIndex();

return \_buffer[i];

}

virtual Type val(unsigned int i, Type val)

{

if (i >= Size)

throw BadIndex();

int oldVal = \_buffer[i];

\_buffer[i] = val;

return oldVal;

}

protected:

Type \_buffer[Size];

};

#endif

#ifndef \_UNIQUE\_ARRAY\_H\_

#define \_UNIQUE\_ARRAY\_H\_

#include "array.h"

class DuplicateValue{};

template <unsigned int Size, class Type>

class UniqueArray : public Array<Size, Type>

{

public:

Type val(unsigned int i)

{

return Array::val(i);

}

virtual Type val(unsigned int i, Type val)

{

for (int ii = 0; ii < Size; ii++)

if (\_buffer[ii] == val)

throw DuplicateValue();

return Array::val(i, val);

}

};

#endif

פתרון שאלה 3:

== 1 ==

A: create a

== 2.1 ==

A: create default\_a

B: create default\_b

== 2.2 ==

A: create b2

B: create b2

== 3 ==

== 3.1 ==

A: foo called for a

== 3.2 ==

B: foo called for default\_b

== 3.3 ==

B: foo called for b2

== 3.4 ==

== 4 ==

A: create copy of b2

B: create copy of b2

A: create copy of default\_a

A: create copy of a

== 4.1 ==

A: foo called for copy of a

== 4.2 ==

A: foo called for copy of default\_a

== 4.3 ==

B: foo called for copy of b2

== 4.4 ==

A: destroy copy of a

A: destroy copy of default\_a

B: destroy copy of b2

A: destroy copy of b2

== 5 ==

B: destroy b2

A: destroy b2

B: destroy default\_b

A: destroy default\_a

A: destroy a

פתרון שאלה 4

חלק א': BASH

produce

#!/bin/bash

cat emails | ./findit |sort -n

findit

#!/bin/bash

while read line; do

line\_split=($line)

student\_=`grep ${line\_split[0]} stud\_ids`

student=($student\_)

if ( [[ ${#student[@]} > 0 ]] ); then

id=${student[0]}

else

id=`echo ${line\_split[0]}|cut -d@ -f1`

fi

echo $id ${line\_split[1]}

done

חלק ב: שאלות הבנה

1. שני המנגנונים עובדים לפי עקרון מקרו: החלפת קוד. הקומפיילר, המקמפל כל קובץ בנפרד, זקוק למימוש הפונקציה באותו קובץ, כדי שיוכל להעתיק ולבצע החלפת קוד. מאחר ולא רוצים (ולא נכון) לממש את הפונקציה במספר קבצי C, שמים את המימוש בקובץ h ועושים include בכל קובץ שבו יש קריאה לפונקציה.
2. catch (exception& e) יכול לתפוש כל חריגה שירשה מ-exception . משתמשים בתכונה הבאה של exception: יש לה פונקציה וירטואלית בשם what, דורסים אותה בכל מחלקה נגזרת

ומממשים אותה כך שתחזיר מחרוזת המתארת את החריגה שקרתה.

1. לסובייקט אין שליטה בכמות המאזינים לו, שיכולה לגדול ולגדול. אם יהיו הרבה מאזינים ובכל שינוי תשלחנה הודעות עדכון לכולם, יתכן עומס רב ופגיעה בביצועים.