060060 4247

המסלול האקדמי
המכללה למינהל

המסלול האקדמי המכללה למינהל

ביה"ס למדעי המחשב

ברקוד נבחן

ת.ז הסטודנט: _____

מספר חדר: _____

מספר נבחן: _____

מספר אסמכתא: _____

מבחן בקורס: תכנות מונחה עצמים

תאריך הבחינה: 08.07.15

שנת הלימודים: תשע"ה, **סמסטר:** ב', **מועד:** א'

משך הבחינה: 3 שעות

שם המרצה/ים:

אליהו חלסצ'י

איגור רוכלין

שם המתרגל/ים:

אילן לופו

חיים שפיר

מבנה הבחינה: הבחינה מורכבת מחלק אחד.

מספר השאלות הכולל בבחינה: 5.

משקל כל שאלה: בצמוד לכל שאלה

הוראות לנבחן:

- אסור השימוש בכל חומר עזר
- יש לענות בגוף השאלון.
- נדרש להחזיר את השאלון.
- לא מצורף נספח לבחינה
- מחברת טיוטה: כן
- מחברת נפרדת לכל שאלה: לא

בהצלחה!!

מפתח בדיקה כללי

א) כל שאלה במבחן בודקת נושא אחר כפי שמפורט בהמשך. אך לאורך כל המבחן, בכל תשובה, אנו בודקים כתיבה נכונה של קוד ב ++C ע"פ הכללים שנלמדו בקורס. להלן הפירוט:

- חוסר ב const או const מיותר – 1 נק'
- העברת פרמטרים \ ערך חזרה בצורה לא נכונה (by ref, by val) – 2 נק'
- פונקציה (גלובאלית) כשנדרשת מתודה או להיפך – 2 נק'
- טעות בהגדרת הרשאות – 1 נק'
- קוד כפול (עבור שאלות שלא הוגדר להן במיוחד ניקוד אחר כמו בשאלה 4) – 3 נק'.

כל קטע קוד במבחן עומד בבדיקה כללית זו באופן בלתי תלוי. כלומר גם אם לדוגמא מישהו טעה ב 3 שאלות שונות בכך שהוא שכח const, אז בכל שאלה תרד לו נקודה אחת. אך אם לדוגמא באותה התשובה הוא טעה 3 פעמים בכך שהוא שכח או הוסיף const מיותר אז תרד לו רק נקודה אחת.

בכל מקרה לא ניתן לאבד יותר נקודות ממה שהשאלה שווה.

ב) בכל תשובה אנו בודקים את המהות. **אם המהות אינה נכונה לא יתקבל ניקוד.** לדוגמא אם המימוש כלל לא נכון \ מכיל יותר מידי טעויות וכו' אז לא יתקבל ניקוד על כך שכתבתם לולאת for או משפט if בסינטקס נכון...

ג) אין אקסטרה ניקוד על קוד מיותר שלא נדרש בשאלה; גם לא נוריד עליו ניקוד. להבדיל מקוד כפול שעליו כן יורדות 3 נק' אז מה שהוגדר במפתח הבדיקה של השאלה.

פתרון המבחן + מפתח בדיקה לכל שאלה

שאלה 1 – ירושה והכלה (35 נק')

הגדרת מחלקה מורכבת משם המחלקה, מה היא יורשת, וה data members שלה. לדוגמא:

```
class B : public A{

        int x,y;
```

במחשב בו ישנם קובצי מוסיקה רבים נהוג לאגד אותם באוספים שונים. לדוגמא ע"פ הז'אנר, האומן, העשור בו הופיע השיר וכו'. אוסף, או באנגלית play list, מורכב מרשימה של קובצי מוסיקה ו\או play-list-ים אחרים.

לדוגמא: האוסף my rock favorites מכיל (בצורה רדודה) 45 פריטים:

- 43 קובצי מוסיקה שונים
- אוסף של ג'ימי הנדריקס שמכיל 5 קובצי מוסיקה
- אוסף של הלהקה U2 שמכיל
  - 8 קובצי מוסיקה
  - אוסף עבור האלבום הראשון שלהם שבו רשומים 2 קובצי מוסיקה.

בסך הכל האוסף My Rock Favorits מכיל (בצורה עמוקה) 58 קבוצי מוסיקה שונים.

ברצוננו להגדיר ספריית מוסיקה עבור קובצי המוסיקה השונים הקיימים במחשב שלנו. עליכם להגדיר את המחלקות הבאות, מותר ואף רצוי להוסיף מחלקות עזר בעת הצורך וע"פ הגיון.

- עבור שדה "שם" יש להשתמש ב *char.

- שם קובץ מלא כולל את הנתיב לקובץ ואת שם הקובץ, לדוגמא "c:\music\blues.mp3".

א. המחלקה Mp3 עבור קובץ בפורמט mp3. למחלקה זו יש
   a. שם הקובץ המלא, גודל (int), תאריך יצירה, תאריך שינוי, תג (char*), וערך bit rate (int)

ב. המחלקה Wave עבור קובץ בפורמט wav. למחלקה זו יש
   a. שם הקובץ המלא, גודל (int), תאריך יצירה, תאריך שינוי, תג (char*), וערך PCM (int)

ג. המחלקה PlayList עבור אוסף מוסיקלי. למחלקה זו יש
   a. שם האוסף
   b. מערך דינאמי שיכול להכיל אובייקטים של קובצי מוסיקה שונים ו\או אוספים מוסיקליים שונים. (ניתן בנוסף לשמור את גודל המערך כמשתנה מסוג int)

תשובה:

```
class Date{...};

class MusicItem{};

class MusicFile : public MusicItem{
	char* fileName;
	int fileSize;
	Date creation, change;
	char* tag;
};

class MP3 : public MusicFile{
	int bitRate;
};

class Wave : public MusicFile{
	int pcm;
};

class PlayList: public MusicItem {
	int size;
	char* name;
	MusicItem** items;

};
```

הסבר:

כבר בסעיפים א' ו ב' אנו רואים בבירור מידע שמשותף ל MP3 ול Wave. לכן את המידע המשותף הזה נגדיר במחלקה (MusicFile) שאותה ירשו המחלקות MP3 ו Wave. נשים לב ש'תאריך' הוא טיפוס (מכיל מידע מורכב), ולא פרמיטיב. לכן נגדיר את המחלקה Date עבור מימוש הטיפוס הזה. לא צריך להגדיר את המשתנים של Date כי לא נדרשנו לכך בשאלה (כמו המגן והחרב במבחן לדוגמא).

עבור ה PlayList של סעיף ג', נשים לב שמבחינה לוגית PlayList אינה סוג של Mp3, Wave או אפילו קובץ מוסיקה; זו תהיה טעות לרשת את אחת המחלקות הללו. ובכל זאת אנו צריכים ליצור מכנה משותף בין ה PlayList לבין ה MusicFile כדי שנוכל ליצור מערך שבו כל תא יכול להכיל או מופע של PlayList או מופע של MusicFile כפי שהסעיף דורש.

לכן, ניצור מחלקה נוספת (MusicItem) שתהווה את המכנה המשותף. המחלקות PlayList ו MusicFile ירשו אותה. מכיוון שגם PlayList וגם MusicFile (ובפרט MP3 ו Wave) הם סוגים של MusicItem, אז *MusicItem יכול להצביע לכל אחד מהטיפוסים הללו. כל שנותר לנו הוא להגדיר בתוך המחלקה PlayList מערך שבו כל תא הוא מסוג *MusicItem כדי לעמוד בהגדרת הסעיף. כל איבר (item) במערך יכול להיות קובץ מוסיקה פשוט או אוסף מורכב.

מפתח בדיקה –

- Mp3 ו Wave יורשות בצורה נכונה מחלקה שמכילה את כל המידע המשותף (MusicFile), ומוסיפות את המידע השונה (bitrate ו pcm בהתאמה) – 13 נקודות.
- Date הוא טיפוס ולא פרמיטיב (כמו *char או int) - 5 נקודות.
- סעיף ג' – 17 נקודות:
  - מחלקה היוצרת מכנה משותף בין MusicFile ל PlayList וירושה נכונה – 5 נק'
  - הגדרה נכונה של מערך בודד העומד בהגדרה – 10 נק'
  - שם האוסף מוגדר במחלקה PlayList – 2 נק'.

הערות נוספות:

- הגדרה של *char name במחלקה MusicItem היא טעות. זה לא מידע שמשותף ל MusicFile ו PlayList. אמנם לכל אחד יש *char משלו, אבל יש לו משמעות שונה ולכן גם תיתכן פונקציונאליות שונה, לכן כל מחלקה מכילה משתנה כזה משלה.
- ירושה וירטואלית של MusicFile אינה נכונה, וגם אינה לא נכונה (לא הורדו נק' למי שהוסיף או למי שלא הוסיף). ירושה וירטואלית צריך להוסיף רק כשצריך – כשברור שייתכנו שילובים ונרצה שהאב הקדמון יופיע רק פעם אחת. במקרה שלנו לא ייתכנו שילובים (קובץ בעל 2 פורמטים שונים). אז באופן ריק לא קרה כלום למי שהוסיף virtual לירושה. אל תסיקו מכך שתמיד צריך להוסיף virtual לירושה לא משנה מה.
- כל מבנה מחלקתי שמאפשר להכניס למערך איברים כמו בדוגמא הבאה צריך לקבל 10 נקודות מתוך סעיף ג' על תשובתו:

```
items[0] = new MP3();
items[1] = new Wave();
items[2] = new PlayList();
```

הערות הבודק \ טעויות נפוצות:

סעיף אחרון - פתרונות בהם כל סוגי הקבצים וגם רשימת השירים ירשו מאותו אבא, ולא היה אב קדמון שיאחד בין רשימת השירים לבין האב של סוגי הקבצים, הורדתי 8 נקודות על ירושה לא נכונה ומשתנים כפולים ומיותרים ברשימת השירים. פתרון של לייצר מחלקה עם שלושה פוינטרים ל MP3,wav,PL ולייצר מערך של המחלקה הזו קיבלו 15- על ירושה לא נכונה ואי עמידה בדרישות, כל מני פתרונות שקשורים במערך מסוג T גם להם ירדו 15 נקודות

שאלה 2 – constructors / destructors (10 נק')

ממשו destructor עבור PlayList. ממשו ע"פ הצורך destructor-ים נוספים במחלקות השונות.

תשובה:

```
virtual ~MusicItem(){}


virtual ~MusicFile(){
	delete[] fileName;
	delete[] tag;
}

virtual ~PlayList(){
	delete[] name;
	for (int i = 0; i < size; i++)
		delete items[i];
	delete[] items;
}
```

הסבר:

כדי לשחרר את PlayList כראוי עלינו לשחרר את שמו, את כל האיברים שבמערך שלו, ואז את המערך עצמו. כל איבר במערך הוא מסוג *MusicItem. קריאה ל delete items[i] תפעיל את ה DTOR של המחלקה MusicItem. אנו רוצים להפעיל את ה DTOR של האובייקט עליו אנו מצביעים בפועל, לכן אנו חייבים להגדיר את ה DTOR של MusicItem כווירטואלי.

גם המחלקות שבהמשך ההיררכיה צריכות להגדיר את ה DTOR שלהן כווירטואלי. לא משום שזו חובת קימפול, אלא מפני שאנו רוצים שמתכנתים אחרים שירשו את מחלקות אלה ידעו בוודאות שמדובר ב DTOR ווירטואלי.

נשים לב שכעת ההפעלה של delete items[i] מפעילה את ה DTOR המתאים. אם זה היה קובץ מוסיקה אז פעל ~MusicFile() ואם זה היה אוסף אז הפעלנו את ~PlayList() על האוסף הזה. כלומר נכנסנו לעומק ומחקנו את שם האוסף, כל האיברים שבמערך של האוסף, ואת המערך עצמו. וכך הלאה בכל פעם שה item הוא PlayList באופן עמוק כל האיברים שלו שוחררו. עשינו זאת בצורה אלגנטית באמצעות virtual DTOR וללא כל typing מסוג כזה או אחר. אגב שאלה זו גם מהווה "חימום" ל 4.

מפתח בדיקה:

- DTOR ווירטואלי ל MusicItem – 4 נק'
- שאר ה DTOR-ים ווירטואלים – 1 נק'
- שחרור נכון של משתני *char - 1 נק'
- שחרור של כל איבר במערך – 4 נק'
-
- חוסר ב DTOR כלשהו (-1) נק'
- חוסר בשחרור משתנה כלשהו (-1) נק'
- שימוש ב typing לצורך השחרור (-2) נק'

הערות הבודק \ טעויות נפוצות:

בדיקה ע"פ מפתח הבדיקה.

שאלה 3 – קבצים (10 נק')

א. ממשו במחלקה Mp3 את המתודה save המקבלת אובייקט מסוג ofstream ובאמצעותו היא שומרת את נתוני ה Mp3.

ב. כמו כן, ממשו את המתודה load המקבלת אובייקט מסוג ifstream שבאמצעותו היא טוענת את נתוני ה Mp3.

ממשו מתודות עזר נוספות במחלקות אחרות במידת הצורך.

תשובה:

```
// inside Date class
void save(ofstream& out) const{...}
void load(ifstream& in){...}

// inside MusicFile class
void saveStr(ofstream& out, const char* str) const{
	int len = strlen(str);
	out.write((char*)&len, sizeof(len));
	out.write(str, len);
}
void loadStr(ifstream& in, char* &str){
	int len;
	in.read((char*)&len, sizeof(len));
```

```
        str = new char[len + 1];
        in.read(str, len);
        str[len] = '\0';
}
virtual void save(ofstream& out) const{
        saveStr(out,fileName);
        out.write((char*)&fileSize, sizeof(fileSize));
        creation.save(out);
        change.save(out);
        saveStr(out, tag);
}
virtual void load(ifstream& in){
        loadStr(in, fileName);
        in.read((char*)&fileSize, sizeof(fileSize));
        creation.load(in);
        change.load(in);
        loadStr(in, tag);
}

// inside Mp3 class
virtual void save(ofstream& out) const{
        MusicFile::save(out);
        out.write((char*)&bitRate, sizeof(bitRate));
}
virtual void load(ifstream& in){
        MusicFile::load(in);
        in.read((char*)&bitRate, sizeof(bitRate));
}
```

הסבר:

מן הסתם את רוב העבודה יש לבצע במחלקה MusicFile בה נמצא רוב המידע; היא האחראית הבלעדית למידע שלה. למחלקה זו יש שני משתנים מסוג *char, שני משתנים מסוג Date, ו int אחד.

כדי למנוע קוד כפול את הטיפול בשמירה \ טעינה של משתנים מסוג *char נעביר למתודות עזר saveStr ו loadStr בהתאמה. השמירה כוללת את שמירת אורך המחרוזת לפני המחרוזת עצמה. ואילו הטעינה קוראת את אורך המחרוזת, מקצה מקום בזיכרון וקוראת את המידע מהקובץ לתוך המקום בזיכרון. שימו לב להעברה של פוינטר by ref כדי שהשינוי על הפרמטר באמת יתבצע עליו ולא על העתק. אפשרות שניה זה פשוט להחזיר פוינטר.

עבור השמירה \ טעינה של תאריך יש להגדיר מתודות מתאימות במחלקה Date, כי מחלקה זו היא האחראית הבלעדית למשתנים שלה. אין צורך לממש זאת במבחן כי לא הגדרנו את תאריך.

המתודה save במחלקה MusicFile:

- שימו לב שהיא וירטואלית. הריי אם יהיה לנו פוינטר של MusicFile שמצביע במקרה למופע של Mp3 אז נרצה להפעיל את ה save של Mp3.
- שימו לב ל const. נרצה הרי לאפשר שמירה של אובייקט קבוע.
  - ראו כיצד זה משפיע על שאר המתודות השמירה שמימשנו.
- שימו לב לסדר שמירת משתני המחלקה ולשימוש החוזר ב saveStr.

המתודה load במחלקה MusicFile:

- גם היא ווירטואלית מאותה הסיבה.
- אותה לא נרצה להפעיל מאובייקט קבוע ולכן אין עליה const.
- שימו לב שסדר הטעינה זהה לסדר השמירה, ולשימוש החוזר ב loadStr.

כל שנותר להשלמת התשובה זה במתודות השמירה והטעינה של MP3 זה להפעיל את השמירה \ טעינה של MusicFile ואז לשמור \ לטעון את משתנה ה bitrate.

שימו לב שבשום פנים ואופן המתודות הללו לא סוגרות (או פותחות) את משתני ה in \ out. מי שרוצה להשתמש במתודות האלו אחראי לפתיחת וסגירת הקבצים. לדוג' פתיחת קובץ, לולאת שמירה של N אובייקטים, ואז סגירת קובץ. אם במתודות אלו נסגור או נפתח קובץ אז האפשרות הזו כבר לא קיימת.

כנ"ל לגבי השמירה \ טעינה של הטיפוס הנשמר. כל אחריות המתודות שלנו זה אך ורק לשמור \ לטעון את המשתנים המוגדרים באותה המחלקה. אם מתכנת אחר רוצה לשמור לפני אובייקט כלשהו את סוגו כי זה מה שהוא רצה בפורמט שהוא בחר אז שיממש; אנו מספקים לו יכולת טעינה ושמירה של האובייקט.

נקודה אחרונה, שימו לב שלא הגדרנו האם השמירה תהיה כטקסט או כמידע בינארי. דיברנו על כך שכשזה לא מוגדר יש לבחור בבינארי כי זה חוסך מקום. במבחן זה לא נוריד נקודות למי שבחר לבצע שמירה כטקסט. אבל בהחלט נוריד נקודות למי שערבב בין טקסט לבינארי באותו הקובץ.

מפתח בדיקה:

- קונסיסטנטיות בין טעינה לשמירה – 4 נק'
- טעינה ושמירה נכונה של הקצאה דינאמית (*char) – 3 נק'
- טעינה ושמירה נכונה של משתנים אחרים – 1 נק'
- Virtual על המתודות + קריאה למתודות של MusicFile – 2 נק'
-
- ערבוב בין טקסט לבינארי - (-5) נק'
- טעויות const – עד (-1) נק'
- קוד כפול – עד (-2) נק'

הערות הבודק \ טעויות נפוצות:

הרחמנות היחידה שלי בבדיקה היתה עבור טעויות שחוזרות על עצמן (עשיתי להן "איחוד").

שאלה 4 – פולימורפיזם + generic algorithm (30 נק')

עליכם לממש את הפונקציות הבאות תוך כדי מינימום קוד כפול. ממשו מתודות עזר מונחות עצמים במחלקות השונות ע"פ הצורך ו\או template function כעזר.

א. בהינתן אובייקט PlayList, הפונקציה count תחזיר את מספר הקבצים הטוטאלי הרשומים באוסף.
ב. בהינתן אובייקט PlayList, הפונקציה size תחזיר את הגודל הטוטאלי של כל הקבצים הרשומים באוסף.
ג. בהינתן אובייקט PlayList, הפונקציה avgBitRate תחזיר את ה bit rate הממוצע של כל קובצי ה mp3 הרשומים באוסף.
ד. בהינתן אובייקט PlayList ואובייקט ofstream, הפונקציה save תשמור בקובץ את נתוני כל האובייקטים שבאוסף.
הערה: אינכם נדרשים לממש את המתודה save עבור המחלקה wave. כן צריך לחשוב קדימה מה יקרה בזמן הטעינה.

תשובה:

הקדמה:

שאלות 4 ו 5 הן השאלות שעלו ברמה ממבחנים קודמים, ובאמת בוחנות את רמת ההבנה ויכולות היישום של החומר שנלמד במהלך הסמסטר. מבחינת ניקוד, שאלות 1-3 מספקות כמעט עובר, וצריך להראות משהו מתוך שאלות 4 או 5 כדי לעבור את המבחן. מי שמצליח לענות על 4 הוא סטודנט טוב מאד, להצליח גם בשאלה 5 זה ההבדל בין טוב מאד למצוין.

שימו לב שבתשובה 4 אין משהו חדש או חומר שלא בחנו עליו בעבר. ההקשר הוא זה שהשתנה. שאלה זו בוחנת את היכולת שלכם להשתמש בכלים שלמדנו בקורס בצורה נכונה וכשצריך.

10 נקודות מתוך ה 30 תקבלו אם **הצלחתם להראות לבודק שהבנתם** שהחלק של לעבור בצורה עמוקה על כל איבריי האוסף זה משהו שחוזר על עצמו בכל סעיף, ולכן זה צריך להיכתב פעם אחת בלבד, ואילו מה לעשות בכל איבר זה משהו שצריך להתקבל "מבחוץ" כפרמרטר – או במילים אחרות כobject function.

10 נקודות נוספות תקבלו במידה והצלחתם ליצור לכך את התשתית (בקוד). כלומר את הפונקציה הגנרית שעוברת בצורה עומקה על כל האיברים שבאוסף ומפעילה "פונקציה" שקבלה כפרמטר על כולם.

יש לכך שתי אופציות עיקריות:

האחת להיעזר במתודה פשוטה במחלקה PlayList בדומה מאד לדרך שבה ה DTOR שלה פעל.

האופציה השנייה היא להיעזר ב iterator ובפעולה ++ שבה הוא תומך (שאלה 5). זו אופציה אלגנטית שחוסכת גם זמן. רק שימו לב שאם בחרתם באופציה השנייה אז כדי לקבל את 10 הנקודות האלה בשאלה 4 עליכם להראות לבודק שאתם יודעים לממש את ה iterator הזה. לא ניתן למשל להגיד ב 4 שהסתמכתם על ה iterator ושלא הספקתם לממש אותו ב 5 בכלל; במקרה זה הבודק לא ראה בקוד שאתם יודעים לענות על השאלה ולכן הניקוד לא יתקבל.

10 הנקודות האחרונות מוענקות למימוש נכון של ה object functions. א' = 2 נק', ב' = 2 נק', ג' = 3 נק', ד' = 3 נק'.

התשובה:

ניצור את התשתית באמצעות הפונקציה הרקורסיבית הבאה:

```
// inside PlayList
template<class func>
friend void applyToAll(PlayList& list,func& f){
   for (int i = 0; i < list.size; i++){
       MusicFile* mf = dynamic_cast<MusicFile*>(list.items[i]);
       if (mf != NULL)         // if it's a type of a MusicFile
          f(list.items[i]);             // apply f on the item
       else                              // else, its a play list
          // so apply f on all the items of the item
          applyToAll(*((PlayList*)(list.items[i])), f);
   }
}
```

הפונקציה applyToAll תקבל אובייקט של PlayList ואיזשהו object function בשם f. שימו לב שלא נשים const בשני הפרמטרים כי את שניהם אנו עתידים אולי לשנות באמצעות applyToAll. היא תעבור על כל האיברים שבמערך items (ניתן לקבל גישה אליו באמצעות getter או שהפונקציה תהיה firend).

אם התא ה i הוא סוג של MusicFile אז נפעיל את f על התא.

שימו לב לשימוש ב dynamic_cast מכיוון שיכולים להיות כמה סוגים של MusicFile. שימוש ב typeid היה מחייב אותנו לשאול האם מדובר ב MP3 או ב Wave. שימו לב שאנו נותנים ל f מצביע מסוג MusicItem. כדי שזה יעבוד, אז בהמשך ה f-ים השונים שנממש יצטרכו לקבל T גנרי או לחילופין לקבל מצביע מסוג MusicItem.

אחרת, התא ה i הוא בוודאות PlayList (לפי המימוש שלנו). לכן עלינו להפעיל את f על כל האיברים שלו. נעשה זא בפשטות ע"י קריאה ל applyToAll על התא שלנו ו f.

שימו לב ש applyToAll רוצה אובייקט מסוג PlayList, ואילו list.items[i] הוא אמנם מצביע לאובייקט מסוג PlayList אך הוא עצמו *MusicItem. נוכל לבצע לו casting ל *PlayList ולתת כפרמטר את ה * של כל זה.

כמו כן, שימו לב ש f מועברת by ref, ולכן זה תמיד יהיה אותו האובייקט f שעובר, ולא העתק.

אופציה ב' היא פשוט לעבור עם iterator על כל האיברים של list ולהפעיל את f רק אם מדובר באובייקט מסוג MusicFile, וזאת תחת ההנחה שמימשתם iterator שה ++ שלו אכן מעמיק.

כל שנותר לנו לעשות הוא לממש את ה object functions. (שימו לב שכל סעיף מהווה רמז לבא אחריו)

במקום להסתבך עם ערכי חזרה, לדוג' אם הם double גם כשצריך int או תמיד להחזיר T שלא ברור מה לעשות אתו וכו'... תזכרו שה object function שלנו מוגדר כמחלקה או struct, וככזה, אין כל בעיה לשמור בתוכו משתנים מסוגים שונים, ולאחר מכן לחלץ את המידע מתוכם.

א.

```
class Count{ // the object function
	int count;
public:
	Count(){ count = 0; }
	void operator()(const MusicItem* mi){
		count++;
	}
	int getCount(){ return count; }
};
// the requested global function
int count(PlayList& pl){
	Count c;
	applyToAll(pl, c);
	return c.getCount();
}
```

ב. הפעם נצטרך לעבור מ MusicItem ל MusicFile כדי לשלוף את גודל הקובצים

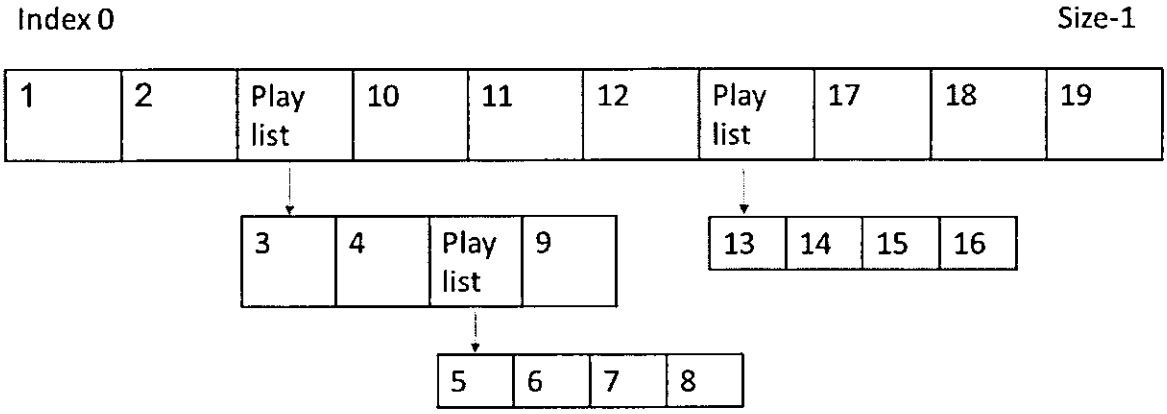
```
class FileSize{ // the object function
	int size;
public:
	FileSize(){ size = 0; }
	void operator()(MusicItem* mi){
		MusicFile* mf = dynamic_cast<MusicFile*>(mi);
		if (mf!=NULL)
			size += mf->getFileSize();
	}
	int getSize(){ return size; }
};
```

שאלה 5 – iterator (15 נק')

בתוך המחלקה PlayList עליכם לממש מחלקת Iterator. ה iterator יכול להצביע אך ורק על קובצי מוסיקה שבאוסף. כלומר, הוא נע בצורה עמוקה על פני כל קובצי המוסיקה שבאוסף. (א) עליכם להגדיר את המחלקה ו (ב) לממש את האופרטור (++) בלבד. אופרטור זה יעביר את ההצבעה לקובץ המוסיקה הבא שברשימה.

כאמור, במקום לדלג על איברים שהם play list, ה iterator "ייכנס פנימה" לתוך ה play list.

לדוגמא, המספרים שבתאים מציינים את סדר תנועת ה iterator כשמופעל האופרטור ++.



<u>**טיפ:**</u> כדאי שבמחלקת ה iterator ננהל רשימה של מצביעים (ולא רק מצביע אחד) כדי שבכל פעם שסיימנו לעבור על תת-אוסף נוכל לחזור לאוסף שהכיל אותו ולעבור לתא הבא.

תשובה:

זו שאלה שדורשת הבנה קצת מתקדמת יותר (או לפתור את המבחן-לדוגמא השני), בגלל המעבר חזרה בין תת-אוסף לאוסף שהכיל אותו. ברור ש iterator עם פוינטר אחד לא יספיק לנו. אין בעיה להזיז אותו פנימה אם הוא אמור להצביע על אובייקט מסוג PlayList. אך כדי לחזור החוצה אנו צריכים לזכור היכן היינו.

ראינו בכיתה דוגמאות לשימוש ב STL. ראינו שניתן להכניס \ להוציא איברים מתחילת \ מסוף רשימה. כמו שהטיפ מרמז, שימוש ברשימה של פוינטרים זה בדיוק מה שאנו צריכים.

לכל רמה יהיה פוינטר משלה. את הפוינטרים האלה נשמור ע"פ הצורך ברשימה. נאפשר לכל פוינטר לנוע אך ורק ימינה. בכל פעם שנרצה להיכנס לרמה עמוקה יותר פשוט נוסיף פוינטר חדש לראש הרשימה, ונדאג שיצביע על המקום נכון. נשים לב שהוא דחק את מי שהיה קודם בראש הרשימה. בכל פעם שנרצה לחזור לרמה בה היינו קודם, פשוט נזרוק את ראש הרשימה. מי שכעת נמצא בראש הרשימה זה הפוינטר שנדחק קודם לכן; הוא שמר לנו היכן הפסקנו.

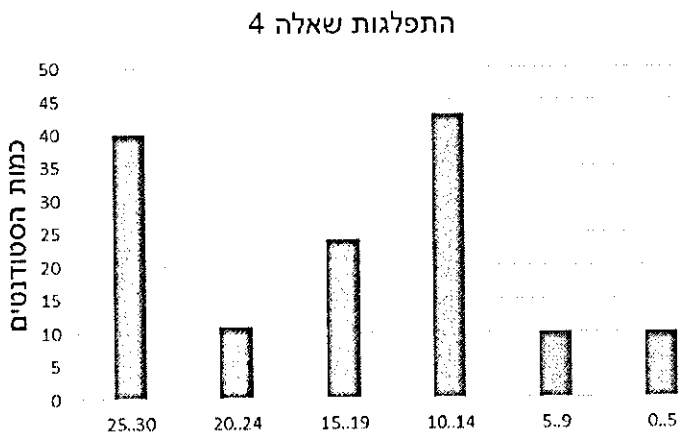
וכעת נדייק יותר. כשנרצה להיכנס לרמה עמוקה יותר, לא נשכח להזיז את הפוינטר הנוכחי ימינה, למרות שנכניס את הפוינטר החדש לרשימה. וזאת מפני שאנו רוצים להיות באיבר הבא כשאנו חוזרים לרמה גבוהה יותר.

נכנס הכי עמוק שאפשר כשמדובר ב PlayList. ניעזר לשם כך במתודה רקורסיבית.

כדי לדעת מתי יש לחזור חזרה לרמה גבוהה יותר, פשוט נבדוק האם אנו בתא האחרון. לשם כך, נגדיר לנו גבול גזרה פשוט – התא האחרון במערך תמיד מצביע ל NULL.

לעולם לא נרשה לפוינטר שבראש הרשימה שלנו להצביע על NULL או על PlayList.

לשם כך ניעזר במתודה רקורסיבית שתמיד תעלה כל עוד אנו מצביעים על NULL.



לא מעט סטודנטים איבדו (רק) 15 נק' אי-שימוש בפונקציה גנרית \ שימוש בקוד חוזר (במקום 20-25 נק' במפתח הבדיקה). אך לא מעט סטודנטים גם הצליחו יפה מאד בשאלה זו. בסך הכל רוב הסטודנטים התמודדו עם השאלה באופן שצפוי למבחן בקורס חובה.

משוב נציגי הסטודנטים לשאלה 4

שאלה 4 הייתה ברמה הגבוהה מהרמה בה תורגל עמנו עד כה.

היא דרשה קו מחשבה שונה ואורכה היה לא יחסי לשאר השאלות במבחן .

במהלך הקורס נלמדו ההבדלים שבין שימוש בפולימורפיזם לתבניות, אך מכאן ועד לבחירת מימוש בין השניים (או שילוב בין השניים) הדרך ארוכה – הנושאים המשולבים לא תורגלו, לא הופיע במטלות או במבחנים לדוגמא, וכמובן שלא במבחנים קודמים .

מסלול ערב מציין, כי נושאי ה-Dynamic Casting ו-Object Functions לא תורגלו עם הכיתה באופן מעשי במהלך הקורס, כמוכן לא ניתנה מטלה מעשית בנושא, ולמרות זאת נושאים אלו היוו חלק משמעותי מהמבחן בשאלה שערכה 30 נקודות.

החלטות:

שאלה זו מאפיינת בעייני את הרמה אליה אנו אומרים להוביל את בוגר הקורס. היא מעין מבוא מקדים ופשוט לקראת תבניות העיצוב בהן אנו עושים שימוש בקורס ההמשך בג'אווה.

בניגוד למשוב הסטודנטים, הבדלים, שילובים, והמוטיבציה לשימוש ב templates ובפולימורפיזם דווקא נלמדו לעומק. אדרבא, נושא זה לא היה סתם עוד נושא בין כל שאר הנושאים שבקורס, אלא יעד(!) שאליו הגענו באמצעות טיפוס בכל המדרגות של הנושאים הבסיסיים יותר.

כאמור, גם היו שאלות בנושא זה במבחנים קודמים, אלא שעד כה, הם לא היו צריכים לזהות הצורך לבדם ולעשות את ההיקש לחומר זה, ודרך משקפת זו אני רואה את המשוב שלהם.

במסגרת שיפור התרגול נכניס את שאלה זו כנושא לשיעור התרגול וכתרגיל בית. כך שטענה כמו זו לא תעלה שוב.

רוב הסטודנטים הצליחו לענות על שאלה זו.

שאלה זו נשארת כבמקור כחלק מהמבחן של מועד א'.

```
// the requested function
int size(PlayList& pl){
	FileSize fs;
	applyToAll(pl, fs);
	return fs.getSize();
}
```

ג. והפעם ל MP3

```
class AVGbitrate{ // the object function
	int count;
	double sum;
public:
	AVGbitrate():sum(0),count(0){}
	void operator()(MusicItem* mi){
		MP3* mp3 = dynamic_cast<MP3*>(mi);
		if (mp3 != NULL){
			count++;
			sum += mp3->getBitrate();
		}
	}
	double getAVGbitrate(){ return sum/count; }
};

// the requested function
double avgBitRate(PlayList& pl){
	AVGbitrate a;
	applyToAll(pl, a);
	return a.getAVGbitrate();
}
```

ד. קצת יותר מתוחכם, כי צריך לדאוג לטעינה. עלינו לשמור גם את כמות הקבצים וגם את הסוג של כל קובץ לפני השמירה שלו. שימו לב שלא נדרשנו לשמור איזה קובץ נמצא באיזה אוסף, אלא רק לשמור את כל נתוני האובייקטים מסוג MusicFile. הנה פתרון פשוט שעושה שימוש במה שכבר יש לנו:

```
class ListSaver{ // the object function
   ofstream* out;
public:
   ListSaver(ofstream* out){ this->out = out; }

   void operator()(MusicItem* mi){
       MusicFile* mf = dynamic_cast<MusicFile*>(mi);
       if (mf != NULL){
          // save the type of the music file as first 2 chars
          char type[2];
          type[0] = typeid(*mf).name()[6]; // skip 'class ', first char
          type[1] = typeid(*mf).name()[7]; // second char
          out->write(type, 2);
          mf->save(*out); // save the music file
          // using the virtual save method we implemented in question 3
       }
   }
};

// the requested function
void save(PlayList& pl, ofstream& out){
	int c = count(pl); // music files count
	out.write((char*)&c, sizeof(c)); // save it
	// deep-save all the objects (and thier types)
	applyToAll(pl, ListSaver(&out));
}
```

הערות הבודק \ טעויות נפוצות:

אי שימוש ב generic function או שימוש בקוד חוזר - הורדת 15 נק' (במקום 20-25 נק')
אי שימוש ברקורסיה (או פונקציה לא נכונה) - הורדת 2 נק' לכל סעיף
חישוב לא נכון של ממוצע בסעיף 3 - הורדת 3 נק'
פונקציה עם קוד לא שלם - הורדת 3 נק'

מתוך ניתוח המבחן - שאלה 4:

שאלה זו היוותה חלק מהעלייה ברמה של המבחן. בשאלה זו הסטודנטים התבקשו לממש 4 פונקציות שונות, כשמה שחזר על עצמו היה המעבר (הרקורסיבי) על כל האיברים שבתוך ה PlayList. מה לעשות עם כל איבר – זה מה שהשתנה, ולכן צריך "להתקבל מבחוץ". הסטודנטים נתבקשו לחסוך בקוד כפול. ולכן מימוש נכון יהיה לכתוב פונקציה שתקבל פונקציה (object function) כפרמטר, ותפעיל אותה על כל האיברים שב PlayList.

המוטיבציה ל object functions + דוגמאות מעשיות ניתנו בשיעורים, ואף באופן מודגש ביחס למחזורים קודמים. כמו כן, בשאלה 6 של המבחן לדוגמא הראשון נעשה שימוש בדיוק בטכניקה זו.

במבחנים קודמים שאלות דומות לשאלה 4 הופיעו בוורסיות מבודדות באופן שהיה ניתן לענות עליהן כמו תוכי (שאלות apply, count_if וכדומה). הפעם הסטודנטים נדרשו גם לזהות את הצורך שמתאים למוטיבציה לשימוש ב object functions, וגם לממש את זה בצורה נכונה. במילים אחרות, שאלה זו דרשה קצת יותר אינטליגנציה – היכולת לפתור בעיות שלא ראו בעבר באמצעות הכלים שלמדו.

חלוקת הנקודות בשאלה זו: 10 אם הראו לבודק **שזיהו את הצורך** בפונקציה גנרית שעוברת על כל האיברים בצורה רקורסיבית ומפעילה "פונקציה" שקבלה כפרמטר, 10 אם מימשו את הפונקציה הזו נכון, ו 10 נק' נוספות אם הם מימשו את ה object functions כהלכה.

השילוב עם פולימורפיזים קיים בעיקר בכותרת של השאלה, ובקריאה למתודות עזר וירטואליות מתוך מחלקות מבלי צורך לדעת על איזה אובייקט בפועל אנו מצביעים.

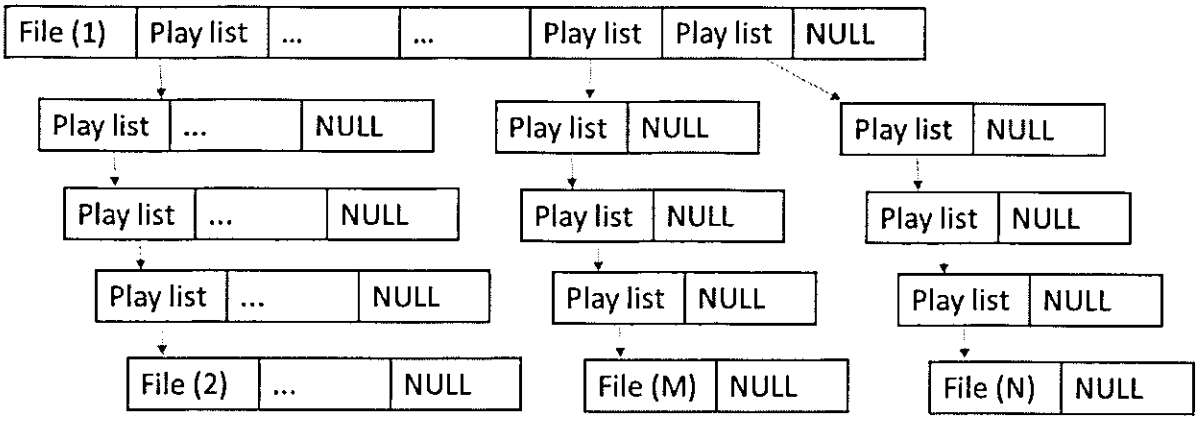
שאלה זו בדקה יכולת

- זיהוי הצורך \ מוטיבציה לשימוש בפונקציה גנרית שמקבלת פונקציה כפרמטר
- מימוש פונקציה רקורסיבית שעוברת על כל האיברים
- הבנה שהטיפוס המשמש אותנו כ object function הוא טיפוס לכל דבר, וככזה ניתן להוסיף לו data members ומתודות עזר ע"פ הצורך.

ממוצע: 17.2/30, סטיית תקן: 7.9

התפלגות הציונים:

דמיינו את **מקרה-הבדיקה** הבא:



אם ה iterator מצביע על File (1) וביצענו ++, נרצה שהוא יצביע על File (2). בכל רמה, אם ++ מוביל אותו ל NULL, נעלה לאיבר הבא ברמה שמעל.

כמו כן, אם ה iterator מצביע על File (M) וביצענו ++, נרצה שהוא יעלה... וירד... ויצביע על File (N). אם כעת נעשה ++ הוא רק יעלה עד הסוף. ולא יהיה לו לאן להמשיך או מה לעשות.

תשובה:

```
// inside PlayList
class iterator{

   list<MusicItem*> pointers; // our stack of pointers

   // goes deepest as possible, until reached a music file
   void goDeep(MusicItem* p){
       PlayList* pl = dynamic_cast<PlayList*>(p);
       if (pl != NULL){ // then it points to a play list
          pointers.front()++; // point to the next item, so we can return to it
          pointers.push_front(pl->items[0]);
          goDeep(pl->items[0]); // go deep!
       }
   }
   // goes highest as possible if at end of an array
   void goHigh(MusicItem* p){
       if (p == NULL && pointers.size() > 1){
          pointers.pop_front();
          goHigh(pointers.front());// go high!
       }
   }
public:
   void operator++(){
       MusicItem* current=pointers.front(); // the pointer at the current level
       current++; // moves right on the array

       if (current == NULL) // at the end of the array
          goHigh(current);

       current = pointers.front();
       if (current != NULL) // maybe on a play list
          goDeep(current);
   }
};
```

מפתח בדיקה:

האופרטור ייבדק על מקרה-הבדיקה שתואר לעיל. יש שם שלושה מקרי קצה. כל מקרה שווה 5 נקודות.

- מ File (1) ל File (2).
- מ File (M) ל File (N).
- מ File(N) ל NULL שבסוף הרמה הגבוהה ביותר.

מתוך ניתוח המבחן – שאלה 5:

שאלה זו אכן כוונה לסטודנטים מצטיינים שיכולים לקחת את החומר הנלמד במקום אחד, וליישם אותו במקום אחר (המבחן הכי אמתי שיש), או לחילופין כוונה לסטודנטים שלא התעצלו לנסות בכוח לפתור את המבחן לדוגמא השני בו היתה שאלה כמעט זהה.

בשיעורים ראינו דוגמאות ל iteratorים בהם יש רק מצביע אחד. כאן, הם התבקשו ליצור iterator שעושה שימוש במחסנית של מצביעים, בהתאמה לעומק (הרקורסיבי) של המבנה.

רק 12/135 סטודנטים השאירו את השאלה הזו ריקה (שלושה נוספים השאירו מבחן ריק). השאר דווקא ניסו לענות. הנתון המטריד הוא ש 55 סטודנטים לא הצליחו לקבל אף נקודה על תשובתם, ורק סטודנט אחד הצליח לקבל את מלוא 15 הנקודות.

המסקנה היא ששאלה זו היתה מעל לרמתם או (וגם) שלא היה להם מספיק זמן לשאלה זו.

משוב נציגי הסטודנטים לשאלה 5

השאלה דרשה ידע מעמיק יותר ב STL-מאשר ברמה התיאורטית שהועברה בקורס. במסלול ערב מעבר לשיעור הבונוס עם חיים שפיר הנושא הנ"ל כמעט ולא תורגל ע"י המתרגלים (וכידוע לא ניתנו לנו מטלות להתנסות תיאורטית בנושא) .

השאלה היחידה שהתקרבה ברמתה אליה הייתה זו שפורסמה כשבוע לפני המבחן עצמו (ב-1.7 לקבוצה של איגור), במבחן לדוגמא 2 – ללא פתרון ותרגול שלה . על כן, לא היו לנו הכלים להתמודד עם השאלה.

הורגש כי מרבית עיסוקה של השאלה הוא בהבנת מבנה הנתונים, קורס שמסלול ערב טרם עבר .

בקורס נחשפנו בעיקר ליישום רשימה מקושרת חד/דו כיוונית עם iterator בסיסי. במבחן נדרשנו ליישום מבנה היררכי הכולל גם פולימורפיזם, ובו איבר במבנה היכול להיות מבנה בעצמו, זוהי קפיצת מדרגה גבוהה מאוד שאינה הוגנת ואינה תואמת את הידע שרכשנו בקורס .

מאחר ורוב הזמן במבחן הנ"ל נלקח לשאלה 4, לא נותר מספיק זמן להתמודדות גם עם שאלה 5.

ההלטות

אני מסכים עם המשוב של הסטודנטים פרט לנושא הקורס מבנה נתונים. כפי שגם הסברנו להם בשיעורי החזרה \ הכנה למבחן, הבחינה אינה על מבני נתונים, אנו לא שואלים על סיבוכיות מקום או זמן, לא דורשים יעילות מעבר לתוכן שנלמד בקורס, ופשוט בוחנים במקצת על היכולת לפתור בעיה שלא ראו באמצעות הכלים שניתנו להם בקורס (או במילה אחרת, אינטליגנציה).

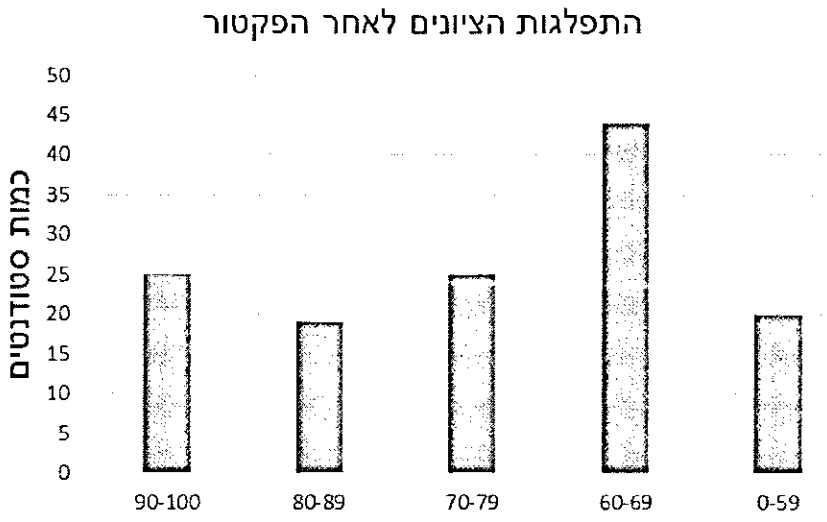
אני שמח שחשפנו את נקודת התורפה הזו, ונצטרך בקורס הבא להכין אותם יותר לקראת התמודדות עם בעיות שלא ראו בעבר באמצעות הכלים שלמדו.

ספציפית לגבי מועד א', אני מבטל את השאלה ומעניק פקטור של 15 נקודות באופן אחיד לכל הנבחנים. מי שקבל ניקוד על שאלה 15 לא מאבד אותו.

לפני הפקטור: ממוצע 57, סטיית תקן 18, מס' נכשלים (ציון קטן מ 60) 66/135.

לאחר הפקטור: ממוצע 72, מס' נכשלים 20/135, ממוצע ללא נכשלים 77.

התפלגות ציונים לאחר הפקטור:



ההתפלגות כעת מתאימה למבחן בקורס חובה.

<u>**הערה:**</u> כולם קבלו 15 נק'. מי שלאחר הפקטור ציונו היה בין 57 ל 59 ציונו עוגל ל 60 בהתאם למדיניות בית הספר. כל ציון נמוך מ 57 לאחר הפקטור נכשל. אני מזכיר ומדגיש שאין טעם לנסות לדוג נקודות כדי להתקרב ל 57 ואז לעבור, היינו רחמנים בבדיקה בלא מעט מקומות; מי שנכשל פשוט נכשל. עליו ללמוד ולהפנים את המסמך הזה ולהצליח בפעם הבאה.

סיכום

שאלה 5 במסגרת הזמן שניתן היתה קשה מידי עבורכם, אך פרט לכך זו תהיה הרמה של המבחן, גם במועד ב' הקרוב, וגם בקורסים הבאים. כן תתבצע התאמה למסגרת הזמן או לרמת השאלות במסגרת הזמן, אך היכולת להשתמש בחומר שלמדתם כדי לפתור בעיות שונות בהחלט תיבחן. מבחן זה, בנוסף לשני המבחנים לדוגמא ישמשו אתכם להתכונן כמו שצריך למבחנים הבאים בקורס זה. נצלו את המידע המפורט שהוענק לכם כאן בפתרון המבחן ובמפתח הבדיקה כדי ללמוד היטב למבחנים הבאים.

בברכת הצלחה(!) בהמשך,

אליהו חלסצ'י.

רכז מקצועות מדעי המחשב.