האוניברסיטה העברית בירושלים

בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

# סדנת תכנות בשפת C ו-++C

# (67312)

# ++C - תרגיל 2

**תאריך הגשה:** יום חמישי, 2021/12/30 בשעה 23:55

תאריך הגשה באיחור בקנס של 10 נקודות: יום שישי, 2021/12/31 בשעה 23:55

**<u>נושאי התרגיל</u>: קוטיינרים, איטרטורים, אקספשנים ואלגוריתמים של STL**

## 1 כללי

בתרגיל זה תממשו עץ AVL עם פונקציית חיפוש ייעודית, שיעזור לכל הסטודנטים לחפש דירה באזור אטרקטיבי. בנוסף, תממשו מחסנית ופונקצית חיפוש גנרית. לבסוף, תבצעו השוואות בין זמני הריצה של דרכי החיפוש השונות, ובין זמני ריצה עם קונטיינרים אחרים. שימו לב שמבנה הנתונים של המחסנית ושל העץ הם נפרדים וניתן לממש אותם ללא תלות אחד בשני. רק בחלק של מדידת הזמנים (7) נצטרך את שניהם על מנת לבצע השוואה.

ניתן להרחיב את ה API ולהוסיף מתודות פרטיות אך אין למחוק או לשנות ממנו שום דבר שהוגדר לכם. לאורך התרגיל ניתן להשתמש בכל ספריה שתראו לנכון, מלבד ספריות שדורשות התקנה מיוחדת, אך שימו לב לאיסורים הספציפיים של כל חלק.

בהצלחה!

## 2 קבצים

<u>הקבצים שעומדים לרשותכן:</u>

קבצי h המגדירים API:

1. קבצי header לכל המחלקות שתידרשו לבנות בתרגיל:
   a. Apartment.h, עבורו תדרשו לממש קובץ cpp תואם.
   b. Stack.h, עבורו תדרשו לממש קובץ cpp תואם.
   c. AVL.h, עבורו תדרשו לממש קובץ cpp תואם.
   d. Find.h, בו תממשו את חלק 6.

קבצי עזר לחלק הבונוס:

2. קובץ MesureTimeHelper.h המכיל פונקציה המפרסרת את קבצי הנקודות לתוך ווקטור של זוגות, ומחלקה שנחוצה עבורכם בשביל מימוש שאלה 3 בחלק 7.
3. קבצים apartments100, apartments1000, apartments10000 המכילים נקודות בפורמט הנכון לצורך הפרסור, כדי שתוכלו למדוד את זמני הריצה של מבני הנתונים שבניתן בתרגיל.
4. קובץ results בו תוכלו למלא את זמני הריצה שקיבלתן.

<u>הקבצים אותם תצטרכו להגיש הם:</u>

a. Apartment.cpp,Apartment.h.
b. Stack.cpp, Stack.h
c. AVL.cpp, AVL.h.
d. Find.h.

ובשביל הבונוס: Bonus.cpp וRESULTS

# 3 מחלקת Apartment

עליכן לממש מחלקת Apartment לפי הAPI הבא:

### 3.1 בנאי

בנאי המקבל זוג של נקודות, כאשר המספר הראשון בצמד הוא קורדיאנטת הX של הדירה ואילו השני הוא קורדינאטת הY.
ניתן להניח את תקינות הקלט. הזוגות ינתנו בצורה של std::pair, כאשר אנו נשתמש ב:
std::pair<double, double>

```
Apartment (const std::pair <double, double>& Coordinates);
```

עוד על השימוש בpair ניתן לקרוא ב[https://en.cppreference.com/w/cpp/utility/pair](https://en.cppreference.com/w/cpp/utility/pair)

### 3.2 Getters

פונקציות get לערך הנקודה בx ובy.

```
double get_x () const;
double get_y () const;
```

### 3.3 אופרטורים של השוואה

שתי דירות זהות אם ערכי הx והy שלהם זהים, עד כדי אפסילון = 0.0001.
יחס הסדר בין הדירות הוא לפי הקרבה שלהן לנקודה `[35.213506 ,31.772425] (הפילבוקס).`
כלומר, אם המרחק של דירה A מהנקודה לעיל הוא 2 והמרחק של דירה B הוא 3 - אז דירה A "קטנה" מדירה B.

```
Apartment operator <(const Apartment& other) const;
Apartment operator >(const Apartment& other) const;
Apartment operator ==(const Apartment& other) const
```

### 3.4 אופרטור הדפסה << Operator

הפורמט בו נדפיס את הדירה יהיה (x,y) וירידת שורה לאחר מכן.

# 4 בניית המחסנית

עליכן לממש מחלקת Stack שמכילה דירות (ניתן להיעזר בכל קונטיינר של stl, **מלבד stack**) לפי הAPI הבא:

## 4.1 בנאי

עליכן לממש שני בנאים:

1. בנאי הבונה מחסנית ריקה:

```
Stack();
```

2. בנאי המקבל וקטור של pair.

כל זוג כזה הוא דירה שנכנסת למחסנית. הזוג הראשון בווקטור הוא הזוג הראשון שיידחף למחסנית.

```
Stack(std::vector<std::pair<double, double>> coordinates);
```

## 4.2 push

מתודה שדוחפת איבר לראש המחסנית:

```
void push(const Apartment& apartment);
```

## 4.3 pop

מתודה שמוחקת את האיבר מראש המחסנית. קריאה למתודה זו על מחסנית ריקה תזרוק שגיאה מסוג out of range עם הודעה אינפורמטיבית לבחירתכן.

```
void pop();
```

## 4.4 empty

מתודה המחזירה אמת אם המחסנית ריקה:

```
bool empty() const;
```

## 4.5 size

מתודה המחזירה כמה איברים יש בתוך המחסנית:

```
size_t size() const;
```

## 4.6 top

מתודות המחזירות את האיבר מראש המחסנית, אך אינן מוציאות אותו ממנה. קריאה לtop על מחסנית ריקה תזרוק שגיאה מסוג out of range עם הודעה אינפורמטיבית לבחירתכן.

```
Apartment& top();
Apartment top() const;
```

## 4.7 תמיכה באיטרטור

בנוסף, בניגוד למחסנית הממומשת ב-stl , למחסנית שלנו יהיה איטרטור.

הערה: באופן עקרוני, stack לא אמורה לאפשר iterator (וכפי שצוין היא ממומשת ללא איטרטור גם ב-STL). חשוב לשים לב לכך, זוהי גם הסיבה מדוע לרוב לא משתמשים ב- stack של STL.

המחסנית אמורה לתמוך באיטרטור (לכל הפחות forward iterator) כך שהאיבר בראש המחסנית הוא האיבר הראשון.
לולאה באמצעות האיטרטור תרוץ על האיבר מראש המחסנית עד לתחתיתה. לדוגמא, אם הכנסתי את 1 ואז 2 למחסנית, האיטרטור יעבור מ2 ל1.

ניתן להעזר ב https://en.cppreference.com/w/cpp/iterator/reverse_iterator

# 5 בניית העץ

עליכן לממש מחלקת AVL לפי הAPI הנתון:

### 5.1 struct Node

כדי לנהל את קודקודי העץ, נעזר בstruct מקונן בתוך מחלקת AVL של קודקוד. מבנה זה מכיל את הדירה המתאימה לקודקוד, את הבן השמאלי והבן הימני של הקודקוד, שניהם מטיפוס Node בעצמם.

שימו לב כי בתוך הAPI של מחלקת ה- AVL מוגדר עבורכן ה- struct בשם Node המכיל את המתודות הבאות:

```
Node *get_left () const;
Node *get_right () const;
const Apartment &get_data () const;
```

**מתודות אלה כבר ממומשות עבורכן, ואסור למחוק/לשנות אותן. מומלץ ואף רצוי להוסיף שדות לNode בכדי להקל עליכן בבניית העץ.**

**שימו לב** על אף שבאופן עקרוני struct זה אינו חלק מהAPI של עץ חיפוש והיה ראוי שיהיה פרטי בתוך מחלקת העץ, אנו נגדיר אותו ציבורי ונגדיר לו API לצורך הטסטים של התרגיל.

מחלקת AVL תכיל שדה שהוא השורש של העץ (מטיפוס פוינטר ל-Node).

ניתן לשנות את הבנאי של Node על מנת לתמוך בשדות נוספים.

הערה: ניתן להניח כי כל האיברים שנכנסים לעץ הם שונים, ומרחקיהם מהנקודה `[35.213506 ,31.772425]` `(הפילבוקס)` שונים.

### 5.2 בנאי

1. בנאי שמאתחל את העץ ללא ערכים.

```
AVL();
```

2. בנאי העתקה.

```
AVL(const AVL& other);
```

(לא לשכוח את חוק השלושה)

3. בנאי המקבל וקטור של pair. כל זוג כזה הוא דירה שנכנסת לעץ, כך שסדר ההכנסה הוא לפי הסדר בו הזוגות מופיעות בווקטור. ניתן להשתמש בinsert על מנת להכניס את הדירות לעץ.

```
AVL(std::vector<std::pair<double, double>> coordinates);
```

### 5.3 get_root

מחזירה את שורש העץ.

```
Node *get_root () const;
```

### 5.4 insert

```
void insert(const Apartment& apartment);
```

**על הפונקציה להכניס את הדירה החדשה לעץ כך שתשמור על החוקיות של העץ, כפי שהיא מתוארת בנספח.**

### 5.5 erase

על הפונקציה למחוק את הדירה (**כפי שמתואר בנספח**) שהיא מקבלת ולשמור על החוקיות של העץ, כפי שהיא מתוארת בנספח. אם הדירה לא נמצאת בעץ, לא יקרה כלום.

```
void erase(const Apartment& apartment);
```

### 5.6 מחלקות מקוננות Iterator ו ConstIterator

יש להגדיר את ה iterator traits ואת כל הפעולות הנדרשות על מנת לתמוך באיטרטור מסוג forward iterator. האיטרטור יזוז בצורה preorder (הסבר בנספח). בפרט, אם יש לנו משתנה a מטיפוס AVL אז a.begin() הוא שורש העץ.

### 5.7 חיפוש בעץ בינארי

יש להגדיר פונקציית חיפוש בעץ בינארי. הפונקציה מחזירה איטרטור לאיבר המתאים לאיבר שחיפשנו. אם אין איבר כזה, מחזירה את end().

```
iterator find (const Apartment &data)
```

```
const_iterator find (const Apartment &data) const
```

זמן הריצה של פונקציית חיפוש זו צריך להיות

O(h)

כאשר h הוא גובה העץ.

### 5.8 Operator << אופרטור הדפסה

יש לממש אופרטור הדפסה שמדפיס את איברי העץ preorder, כאשר כל איבר מודפס באמצעות אופרטור ההדפסה של דירה.

## 6 Find

עליכן לממש פונקציית find (בקובץ הh) עם החתימה הבאה:

```
template< class InputIt >
```

```
InputIt find( InputIt first, InputIt last, const Apartment& value ){
```

פונקציה זו היא פונקציה שעושה שימוש ב- Template, כפי שתלמדו בהרצאה. בנתיים, שימו לב שאתן ממשות את הפונקציה בקובץ h, ומתייחסות first ולast כinput איטרטור ועל כן מותר להשתמש בתכונות שלמדתן שאיטרטורים מסוג זה מקיימות.

**בחלק זה אין לעשות שימוש בשום אלגוריתם מstl.**

## 7 בונוס - מדידת זמנים וביצועים- 10 נקודות בונוס

בתוך קובץ cpp בשם Bonus עליכן לקלוט קבצים שמכניסים 100, 10000 ואז 10000 דירות באמצעות פונקציה `xy_from_file` שתינתן לכן בקובץ MesureTimeHelper.h .

חשוב!! לא להגיש פונקציית main בקובץ Bonus.cpp, כלומר תוכלו לכתוב את כל מדידת הזמנים בפונקציה (ולקרוא לה mainמ חיצוני אותו לא תגישו).

עליכן לבנות מופע של המחלקה דירה עם הנ"צ הבא: `31.81428051893798,35.18577781093502` נקרא לה דירה X.

עליכן להכניס את הדירות למחסנית ולעץ AVL . בקובץ RESULTS עליכן לתעד כמה זמן לקחו שלושת הפעולות הבאות (עבור כל קובץ של דירות, בננו שניות):

1. מחסנית:
   a. הכנסת כל הדירות למחסנית.
   b. חיפוש דירה X במחסנית באמצעות פונקציית find הכללית.

2. עץ AVL:
   a. הכנסת כל הדירות לעץ.
   b. חיפוש דירה X בעץ באמצעות פונקציית find של העץ.
   c. חיפוש דירה X בעץ באמצעות פונקציית find הכללית.
3. Unsorted set:
   a. הכנסת כל הדירות לunsorted set.
   b. חיפוש דירה X בunsorted set באמצעות find הכללית שבניתם.
   c. חיפוש דירה X בunsorted set באמצעות find שלה.

שימו לב שגם עבור חלק זה יש להשתמש בMesureTimeHelper.h. לאחר שתייבאו את הקובץ הזה, נגדיר unorder set של דירות כך:

```
std::unordered_set<Apartment, MyHashFunction> a;
```

### 7.1 מדידת זמן בC++

על מנת למדוד זמן בננו שניות נשתמש בספרייה:

std::chrono::

נשתמש ב

```
auto t2 = std::chrono::high_resolution_clock::now();
```

על מנת למדוד מה השעה המדויקת לפני תחילת הפעולה אותה נרצה למדוד ונשתמש באותה הפונקציה שוב בסיום הפעולה.
כדי לחשב את ההפרש בננו שניות נשתמש בשורה הבאה:

```
std::chrono::duration_cast<std::chrono::nanoseconds>(t2-t1).count();
```

כאשר במקרה הזה t1 הוא תחילת המדידה וt2 הוא סיומה.

## 8 נהלי הגשה

**8.1** הגשת התרגיל תתבצע בעזרת ה- git והקבצים שיש להגיש הם:

1. Apartment.cpp.
2. Apartment.h.
3. Stack.cpp.
4. Stack.h.
5. AVL.cpp
6. AVL.h.
7. Find.h.

בשביל הבונוס: Bonus.cpp וRESULTS .
הגשה של כל קובץ אחר תוביל לכישלון בתרגיל.

**8.2** זיכרו לוודא שהתרגיל עובר קומפילציה במחשבי בית הספר ללא שגיאות ואזהרות. בנוסף, נזכיר שיש לתעדף פונקציות ותכונות של C++ על פני אלו של C. למשל, נשתמש ב־new ו־delete על פני malloc ו־free.

**8.3** אנא וודאו כי התרגיל עובר את הפריסבמיט ללא שגיאות או אזהרות. בכדי להריץ אותו על התרגיל שלכן על מחשבי בית הספר, עליכן לייצר קובץ tar המכיל את הקבצים שיש להגיש, ואז יש להריץ את הפקודה הבאה:
~labcc/presubmit/ex6/run <path to ex6.tar>

בהצלחה!!

# 9 נספח - AVL

## 9.1 הכנסה בעץ בינארי

אם העץ ריק, נכניס את הערך החדש בתור השורש. אחרת, נכניס אותו לפי חוקיות העץ.

## 9.2 מחיקה בעץ בינארי

נחלק למקרים:

1. הקודקוד שימחק הוא עלה: פשוט נסיר אותו מהעץ.
2. לקודקוד שימחק יש רק בן אחד: נעתיק את הבן לאב ונמחק את הבן.
3. לקודקוד שימחק יש שני ילדים: נמצא את העוקב של הקודקוד בעץ. נעתיק את התוכן שלו לקודקוד ונמחק את העוקב.

   (מיהו העוקב? האיבר המינימלי בתת העץ הימני של הקודקוד)

לאחר מכן נוודא שאנחנו שומרות על חוקיות העץ.

## 9.3 חוקיות העץ

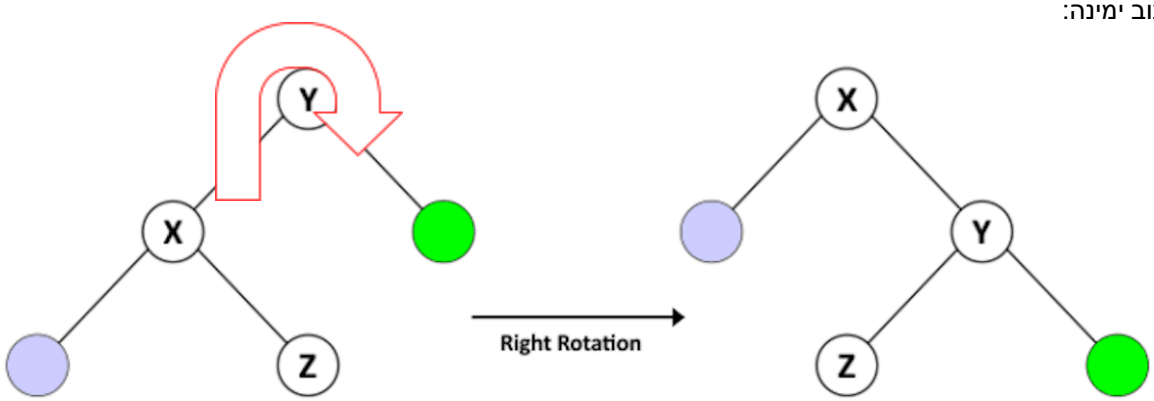
עץ AVL הוא עץ AVL חוקי אם:

1. תת העץ השמאלי של כל קודקוד מכיל רק קודקודים קטנים יותר מהקודקוד הנוכחי.
2. תת העץ הימני של כל קודקוד מכיל רק קודקודים גדולים יותר מהקודקוד הנוכחי.
3. תתי העצים השמאליים והימניים צריכים גם הם לקיים את התכונות לעיל.
4. הערך המוחלט של ההפרש בין הגבהים של תת העץ הימני והשמאלי הינו לכל יותר 1.

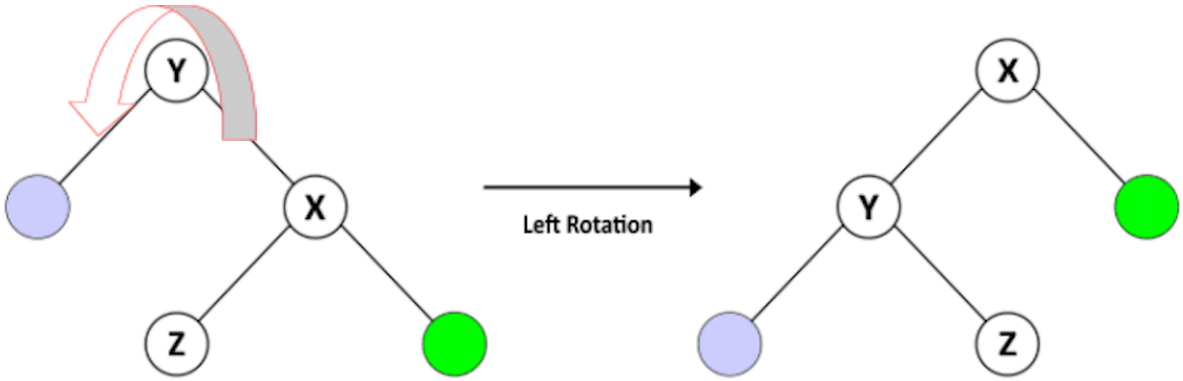
לכן, אם הכנסת דירה חדשה על פי הכללים 1-3 מפרה את כלל 4, נצטרך לבצע איזון, או לחילופין, אם מחיקה של קודקוד מעץ בינארי יוצרת הפרה, נבצע גם כן איזון.

האיזון יעשה באמצעות סיבוב. ישנם שני סוגי סיבובים (שימו לב שתכונת העץ הבינארי נשמרת):

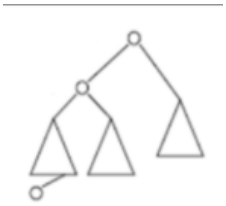
סיבוב ימינה:




סיבוב שמאלה:

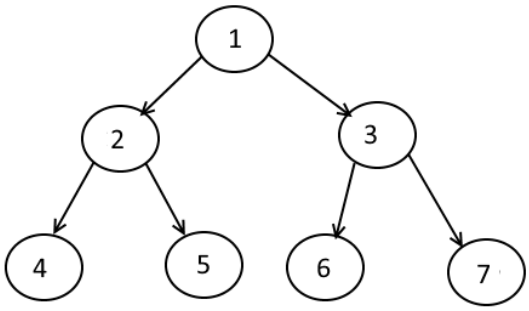



לאחר שנבין איזו סוג הפרה קרתה, נבצע את האיזונים לפי הטבלה הבאה, כאשר גורם האיזון הוא הגובה של הבן שמאלי פחות הגובה של הבן הימני:

| סוג ההפרה | LL | RR | LR | RL |
|---|---|---|---|---|
| |  |  |  |  |
| מה מאפיין אותה? | 1. גורם האיזון בשורש תת העץ הוא 2<br>2. גורם האיזון בבן השמאלי של השורש הוא 1 | 1. גורם האיזון בשורש תת העץ הוא -2<br>2. גורם האיזון בבן הימני של השורש הוא -1 | 1. גורם האיזון בשורש תת העץ הוא 2<br>2. גורם האיזון בבן השמאלי של השורש הוא -1 | 1. גורם האיזון בשורש תת העץ הוא -2<br>2. גורם האיזון בבן הימני של השורש הוא 1 |
| איזו רוטציה נבצע? | רוטציה ימינה על השורש | רוטציה שמאלה על השורש | 1. רוטציה שמאלה על הבן השמאלי של השורש<br>2. רוטציה ימינה על השורש | 1. רוטציה ימינה על הבן הימני של השורש<br>2. רוטציה שמאלה על השורש |

## 10 נספח - Preorder traversal

במעבר מסוג preorder נבקר תחילה בקודקוד הנוכחי, לאחר מכן נבקר בבן השמאלי (כולל כל תת העץ שלו), ולאחר מכן נבקר בבן הימני (כולל כל תת העץ שלו).




Preorder Traversal: 1 2 4 5 3 6 7