האוניברסיטה העברית בירושלים
בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות בשפת C ו-++C (67312) תרגיל 7

תאריך הגשה : 13 לינואר 2022, בשעה 23:55.

הגשה מוקדמת מזכה בבונוס והגשה באיחור בקנס – פירוט בהמשך ההוראות.

1 רקע

1.1 אמל"ק

בתרגיל זה תדרשו לממש container חדש ויעיל, הזהה ל- std::vector<T> מבחינת התנהגות, אך חסכוני יותר בזמני ריצה. תממשו מבנה נתונים המתנהג כמו ווקטור, אך מממש שיטה יעילה לניהול זיכרון המנצלת את יתרונותיהם של המחסנית (stack) והערימה (heap) וממזערת את חסרונותיהם.

קראו היטב את ההוראות במסמך, בפרט את אלו הנוגעות לטיפוסים מ-STL ולנושאי יעילות.

1.2 זיכרון סטטי וזיכרון דינאמי – תזכורת

במהלך הקורס למדנו דרכים לשמירת ערכים ומבני נתונים בזיכרון, ודיברנו על שני מקטעים רלוונטיים: ה-stack וה-heap. הבחירה להשתמש בזיכרון סטטי או דינמי תלויה בסיטואציה, כשלכל כלי יתרונות וחסרונות, ולנו האחריות להשתמש בכלים העומדים לרשותנו בתבונה:

	זיכרון סטטי	זיכרון דינאמי
מקטע רלוונטי	Stack	Heap
זמינות	זמין לנו כברירת מחדל בכל פונקציה, לכל פונקציה יש קטע ב-stack ששייך לה	עומד לרשותנו רק כשנבקש זאת במפורש, באמצעות בקשה להקצאת זיכרון דינמי
מהירות	מהיר	איטי וניהול מורכב יותר
זמינות לאורך זמן	זמין לזמן קצר	זמין לזמן ארוך
אופן שחרור	אוטומטי בסיום השמורה (scope). גישה לזיכרון אחרי שחרורו תיחשב כקריאה לא חוקית	לאחר בקשה מפורשת ממערכת ההפעלה לשחרר באופן מפורש את הזיכרון, אחרת תיווצר דליפות זיכרון
מגבלת גודל	ל-stack גודל מקסימלי שלא ניתן לחצות (למשל, גודל המחסנית במחשבים עם Windows הוא MB1). לכן חייבים לדעת מראש מה הגודל המקסימלי של הקלט.	כמות גדולה יותר של זיכרון. ניתן לנצל יתרון זה בייחוד במקרים בהם איננו יודעים מהו גודל הקלט
מתי נקבע גודל הזיכרון	זמן קומפילציה	זמן ריצה

<u>הערה</u>: כשעסקנו בשפת C למדנו על Variable Length Array (VLA) – מערך בגודל שאינו קבוע (נקבע בזמן ריצה ולא קומפילציה) ומוקצה על ה-stack. לאור הבעייתיות המובנית (הקצאה על ה-stack, שמוגבל בזיכרון) – השימוש בו לא מומלץ ואף אסור במסגרת הקורס.

1.3 הגדרות וסימונים שנשתמש בהם במסמך

1. יהי $\boldsymbol{size} \in \mathbb{N} \cup \{0\}$ כמות האיברים **הנוכחית** בוקטור (לפני הוספה / הסרה של איברים).
2. תהי $\boldsymbol{k} \in \mathbb{N}$ כמות האיברים בפעולה (שנרצה להוסיף בפעולת הוספה או להסיר בפעולת הסרה).
3. יהי $\boldsymbol{C} \in \mathbb{N} \cup \{0\}$ קבוע המסמל את הזיכרון **הסטטי** המקסימלי של הווקטור ($\boldsymbol{StaticCapacity}$).

2 זיכרון סטטי וזיכרון דינמי

2.1 הגדרת טיפוס הנתונים Variable Length Vector

נגדיר את ה-container "וקטור באורך משתנה" vl_vector להיות טיפוס נתונים **גנרי** הפועל על אלמנטים מסוג T ובעל קיבולת סטטית C. יממש API דומה ל-std::vector, אך ישתמש ב-stack וב-heap לאחסון.

2.2 אלגוריתם נאיבי לאחסון

vl_vector יפעל באמצעות האלגוריתם הנאיבי הבא כדי "לתמרן" ביעילות בין שימוש ב-stack וב-heap:

- <u>הצהרה</u>: הווקטור יקבל שני פרמטרים **גנריים**: את טיפוס הנתונים שאותם הוא מאחסן T, ואת C שיסמן את מספר איברים המקסימלי שהווקטור יכול להכיל באופן **סטטי**. מטעמי יעילות, נדרוש כי כל מופע של vl_vector יתפוס C ערכים **בדיוק** ב-stack.

- <u>הוספת איבר(ים) לווקטור</u>:[1]
  - **אם $size + k \leq C$:** הוספת k איברים לא תחצה את C ולכן k הערכים החדשים ישמרו ב-stack.
  - **אם $size + k > C$:** לא נוכל לשמור את כל k האיברים בזיכרון הסטטי, ולכן הווקטור יפסיק **באופן גורף** להשתמש בזיכרון סטטי ויעבור להשתמש בזיכרון דינמי. לשם כך הווקטור יקצה את כמות הזיכרון הנדרשת (כמפורט בחלק הבא) **ויעתיק אליו** את כל הערכים שעד כה נשמרו על ה-stack (לא ניתן להימנע מהעתקה[2]). לבסוף, גם k האיברים הנוספים יישמרו בזיכרון הדינמי.

- <u>הסרת איבר(ים) מהווקטור</u>:
  - **אם $size - k > C$:** הסרת k איברים לא תגיע ל-C, ולכן k הערכים יוסרו מה-heap (את שאר הערכים הווקטור ימשיך להחזיק ב-heap).
  - **אם $size - k \leq C$:** נעתיק חזרה את הערכים ל-stack, נמחק את הזיכרון הדינמי ונחזור להשתמש בזיכרון הסטטי.

כשאין צורך לעבור מזיכרון סטטי לזיכרון דינמי ולהיפך, הפעולות ימומשו עם הזיכרון הרלוונטי. למשל הסרת איברים כשהווקטור נמצא בזיכרון הסטטי – האיברים יוסרו מהזיכרון הסטטי; הוספת איברים כשהווקטור נמצא בזיכרון הדינמי – נוסיף אותם לזיכרון הדינמי, בהתאם לפונקציית הקיבולת שלהלן.

2.3 קיבולת הווקטור

לווקטור תהיה פונקציית קיבולת (capacity) כמו ל-std::vector, המתארת את המספר המקסימלי של האיברים שהוא יכול להכיל בכל רגע נתון. נגדיר את פונקציית הקיבולת: $cap_C: \mathbb{N} \cup \{0\} \times \mathbb{N} \to \mathbb{N}$.
בהינתן C קבוע, cap_C תחזיר את הקיבולת המקסימלית של הווקטור, בהתאם לקלט: $size$ ו-k.

לנגד עיננו שתי מטרות: מצד אחד, נרצה לשמור על זמני ריצה טובים ככול האפשר. נרצה שפעולות הגישה, ההוספה וההסרה יפעלו כולן ב-$O(1)$. מהצד השני, לא נרצה להקצות יותר מדי מקום, שיתבזבז לשווא.

כשמדובר בזיכרון **סטטי** ($size + k \leq C$), הקיבולת של הווקטור היא C תמיד. מה הקיבולת כשנחצה את C ונעבור להשתמש בזיכרון **דינמי**? ניסיון נאיבי יהיה להגדיל את הווקטור כל פעם ב-k איברים. למשל בהוספת איבר חדש יחיד ($k = 1$), נקצה את כל הווקטור מחדש עם $(size + 1) \cdot sizeof(T)$ בייטים ונעתיק לתוכו את איבריו של הווקטור הישן. גישה זו פועלת בזמן ריצה של $O(n)$ ולכן אינה מתאימה[3].

---

[1] ראינו שב-std::vector ניתן לקרוא לפעולת insert כך שתוסיף איבר יחיד או מספר איברים (בעזרת iterator).
[2] למעוניינים בפירוט – ראו מסמך העשרה במודל.
[3] הוכחת מתמטית – ראו מסמך העשרה במודל.

להלן הפתרון[4] אותו תצטרכו לממש :

נגדיר את פונקציית הקיבולת כך :

$$cap_C(size,k)=\begin{cases} C & size+k\le C \\ \left\lfloor \frac{3\cdot(size+k)}{2} \right\rfloor & otherwise \end{cases}$$

מטעמי יעילות, בעבודה זיכרון דינמי נאפשר רק **הגדלה** של הקיבולת ולא הקטנה. אם לאחר הסרה, cap_C תראה שיש להשתמש בזיכרון דינמי, אך ה-capacity הנדרש קטן מזה שיש כעת, לא נצמצם את הווקטור[5].

לסיכום:

- יש להשתמש בזיכרון סטטי **כל עוד** כמות האיברים בווקטור אינה **חוצה** את C.
- יש להשתמש בזיכרון דינמי **כל עוד** כמות האיברים **חצתה** את C.
- יש לתמוך במעבר מזיכרון סטטי לזיכרון דינמי, **ולהיפך**.
- יש לעמוד בחסם של $O(1)$ לשיעורין לפעולת הגישה, ההוספה/ההסרה לסוף/מסוף הווקטור.
- זכרו שאין מנוס מהעתקת האיברים בכל הגדלה / הקטנה.
- נחשב את cap_C אך ורק בפעולות ההוספה וההסרה. **עם זאת, נזכיר:** בעבודה עם זיכרון דינמי, קיבולת הווקטור יכולה **רק** לגדול. לכן בעת שימוש בזיכרון דינמי, נחשב את הקיבולת מחדש רק בהגדלת הווקטור (רק כתוצאה מהוספת איבר(ים)).
- בהסרת איברים עד לערך הקטן או השווה ל-C ($size-k\le C$) חוזרים לעשות שימוש בזיכרון הסטטי, וה-capacity הדינמי "מתאפס". כשנגדיל שוב עד $size+k>C$ לא נחזור להשתמש ב-capacity הישן, אלא נחשב את הערך הנכון מחדש עם cap_C.
- **נדגיש** : המימוש שלכם **חייב** להשתמש בהגדרת cap_C לחישוב קיבולת הווקטור בכל עת. שינוי גודל הווקטור בצורה שונה או החזרת ערכים לא תואמים, יפגעו **משמעותית** בציון.

דוגמה למעברים בין ערכי ה-capacity וסוגי הזיכרון (סטטי/דינמי) מופיעה בסעיף 5.3 בהמשך התרגיל.

3 המחלקה vl_vector

הנכם נדרשים לממש בקובץ vl_vector.h את המחלקה הגנרית vl_vector.
מבנה הנתונים שלכם ישמור ערכים מסוג T ועם קיבולת סטטית StaticCapacity (שניהם משתנים גנריים שהמחלקה מקבלת).
ל-StaticCapacity נגדיר ערך ברירת מחדל של 16.

3.1 עליכם לתמוך ב-API הבא:

[4] לנימוקים בבסיס הגדרה זו של cap_C – ראו מסמך העשרה במודל.
[5] הפתרון לא בהכרח יעיל. כדי לפתור זאת $vector$ :: std מציע את הפעולה shrink_to_fit. אינכם נדרשים לממש אותה.

	התיאור	הערות	זמן ריצה
פעולות מחזור החיים של האובייקט			
Default Constructor	בנאי שמאתחל vl_vector ריק.		$O(1)$
Copy Constructor	מימוש של בנאי העתקה.		$O(n)$ n הוא מספר האיברים בווקטור המועתק.
Sequence based Constructor	בנאי המקבל Input Iterator (מקטע [first,last) של ערכי T ושומר את הערכים בוקטור. החתימה המלאה בהמשך.	הגדלת הוקטור, אם יש בה צורך – תיעשה עבור כל n האיברים יחדיו, בפעם אחת, ולא עבור כל איבר לחוד.	$O(n)$ n הוא מספר האיברים ב- [first,last).
Single-value initialized constructor	בנאי המקבל $count \in \mathbb{N} \cup \{0\}$ כמות ואיבר v כלשהוא מסוג T. הבנאי מאתחל את הווקטור עם count איברים בעלי הערך v.	יש לחשב את ההכנסה של האיברים כרצף ולא כהכנסה של איבר איבר.	$O(count)$
Destructor	מימוש Destructor.		
פעולות			
size	פעולה המחזירה את כמות איברים הנוכחית בווקטור.	ערך החזרה מטיפוס size_t.	$O(1)$
capacity	פעולה המחזירה את קיבולת הווקטור הנוכחית.	ערך החזרה מטיפוס size_t. לא צריך לחשב בפונקציה זו את cap_C.	$O(1)$
empty	פעולה המחזירה אם הווקטור ריק.	ערך החזרה bool.	$O(1)$
at	פעולה המקבלת אינדקס ומחזירה את הערך המשויך לו בווקטור.	הפעולה תזרוק חריגה אם האינדקס שגוי.	$O(1)$
push_back	הפעולה מקבלת איבר ומוסיפה אותו לסוף הווקטור.	הפעולה אינה מחזירה ערך.	$O(1)$ $(amortized)$[6]
insert (1)	פעולה המקבלת איטרטור המצביע לאיבר מסוים בווקטור (position), ואיבר חדש. הפעולה תוסיף את האיבר החדש **לפני** ה-position (משמאל ל-position).	הפעולה תחזיר איטרטור המצביע לאיבר **החדש** (לאיבר שנוסף כעת).	$O(n)$ כאשר n הוא כמות האיברים בווקטור (size).
insert (2)	פעולה המקבלת איטרטור המצביע לאיבר מסוים בווקטור (position), ו-2 משתנים המייצגים Input Iterator למקטע[first,last). הפעולה תוסיף את ערכי האיטרטור **לפני** ה-position.	הפעולה תחזיר איטרטור שמצביע לאיבר **<u>הראשון</u>** מרצף האיברים החדשים. כדי להסיק איך להגדיר את first ו-last היעזרו בחתימה של ה-	$O(n)$ כאשר n הוא כמות האיברים בווקטור, בחיבור כמות האיברים במקטע [first,last)

[6] זמן ריצה לשיעורין. ראו: https://bit.ly/3jSVAsQ.

		sequence based constructor שבהמשך.	
pop_back	הפעולה מסירה את האיבר האחרון מהווקטור ואינה מחזירה ערך.	אם size=0 יש לעצור מבלי לזרוק חריגה.	$O(1)$ $(amortized)$
erase (1)	הפעולה מקבלת איטרטור של הווקטור ומסירה את האיבר שהוא מצביע עליו.	הפעולה תחזיר איטרטור לאיבר **שמימין** לאיבר שהוסר.	$O(n)$ כאשר n הוא כמות האיברים בווקטור.
erase (2)	הפעולה מקבלת 2 משתנים המייצגים איטרטור של מופע ה - vl_vector, למקטע [first,last). הפעולה תסיר את הערכים שבמקטע מהווקטור.	הפעולה תחזיר איטרטור לאיבר **שמימין** לאיברים שהוסרו.	$O(n)$ כאשר n הוא מספר האיברים בווקטור.
clear	הפעולה מסירה את כל איברי הווקטור.		$O(n)$ כאשר n הוא מספר האיברים בווקטור.
data	הפעולה מחזירה **מצביע** למשתנה שמחזיק כרגע את המידע.	הפעולה תחזיר מצביע למשתנה שמחזיק את האיברים ב-stack או ב-heap, בהתאם למצב הנוכחי של הווקטור.	$O(1)$
contains	הפעולה מקבלת משתנה מסוג T ומחזירה ערך בוליאני האם הערך נמצאת בוקטור.		$O(n)$ כאשר n הוא מספר האיברים בווקטור.
Iterator Support	על המחלקה vl_vector **לתמוך** ב- iterator (**לרבות typedefs**) בהתאם לשמות הסטנדרטים של C++.	עליכם **לתמוך** ב- Random Access Iterator (const ו-non const).	על כל הפעולות הנדרשות לעמוד בזמן ריצה של $O(1)$.
Reverse Iterator Support	על המחלקה vl_vector **לתמוך** ב- reverse iterator (**לרבות typedefs**) בהתאם לשמות הסטנדרטים של C++.[7]	עליכם **לתמוך** ב- Random Access Iterator (const ו-non const).	
אופרטורים			
השמה	תמיכה באופרטור ההשמה (=).		
subscript	תמיכה באופרטור [ ].	האופרטור יקבל אינדקס ויחזיר את הערך המשויך לו. **אין לזרוק חריגה במקרה זה.**	$O(1)$
השוואה	תמיכה באופרטורים ==, =!	שני ווקטורים שווים אחד לשני אם ורק אם איבריהם שווים ומופיעים בסדר זהה.	

[7] מומלץ לקרוא את המקור הבא: https://en.cppreference.com/w/cpp/iterator/reverse_iterator.

3.2 דגשים, הבהרות, הנחיות והנחות כלליות:

- החתימה ל-Sequence based Constructor (רלוונטית עם שינויים ל-insert 2 ול-erase 2) היא:
```
template<class InputIterator>
vl_vector(InputIterator first, InputIterator last);
```
- על המחלקה להיות **גנרית.** הפרמטר הגנרי הראשון הוא טיפוס הנתונים שהמחלקה מאחסנת (T). הערך הגנרי השני הוא הקיבולת המקסימלית שניתן לאחסן באופן סטטי (StaticCapacity או C).
- **ניתן להניח** כי מופעים מסוג T תומכים ב-operator==, operator= וכן כי יש למופעי T בנאי דיפולטיבי ובנאי העתקה.
- **בפתרונכם אינכם רשאים להשתמש באף container של STL**, אך ניתן להשתמש בכל אלגוריתם של STL. בפרט, אין לעשות שימוש ב-std::array, std::vector, ו-std::list. **שימוש ב-containers יגרור בהכרח ציון 0**. כמו כן, לא ניתן להשתמש בספריות חיצוניות.
- בחלק הנוגע ל-iterators שמות הפונקציות יהיו **באותיות קטנות**, כמקובל ב-STL.
- בעת מימוש ה-API הנ"ל, עליכם ליישם את העקרונות שנלמדו בקורס באשר לערכים קבועים (constants) ומשתני ייחוס (references). **שימוש בקונבנציות אלו הוא חלק אינטגרלי מהתרגיל**. עיקרון זה נכון בפרט גם לגבי מימוש ה-iterator.
- לפני שתיגשו לחיבור הפתרון, חישבו על **כל** הכלים שרכשתם בקורס. בפרט, כשאתם שוקלים האם האופציה X מתאימה למימוש – חישבו בין היתר איזה תכונות יש לה? היכן היא מוקצית? מה היתרונות שלה? מה היא דורשת מכם מבחינת מימוש? **שימוש נכון בכלים שונים שלמדנו יקצר את מרבית הפונקציות לאורך של כמה שורות בלבד**, ויאפשר לכם לקבל "במתנה" חלק נכבד מהמימוש.

4 נהלי הגשה

- עליכם להגיש את הקובץ vl_vector.h בלבד. בדומה לכל תרגילי הקורס יש להגיש דרך git.
- וודאו שתרגילכם עובר קומפילציה במחשבי בית הספר ללא שגיאות ואזהרות, כנגד מהדר בתקינה שנקבעה בקורס (C++14). אזהרות יביאו בהכרח לגריעת ניקוד (בהתאם לחומרת האזהרות). תרגיל שאינו עובר הידור, ינוקד בציון 0. יש לתעדף פונקציות ותכונות של C++ על פני אלו של C. למשל, נעדיף להשתמש ב-new ו-delete על פני malloc ו-free.
- הקצאת זיכרון דינמית מחייבת שחרור זיכרון. היעזרו ב-valgrind כדי לאתר דליפות זיכרון. **עליכם למנוע בכל מחיר דליפות זיכרון מה-container שלכם**. דליפות זיכרון יאבדו ניקוד **משמעותי**.
- וודאו שתרגילכם עובר את ה-Pre-submission Script ללא שגיאות או אזהרות. הקובץ זמין בנתיב:
```
~labcc/presubmit/ex7 <path_to_sibmission>
```
- לתרגיל זה לא ניתן פתרון בית ספר. כחלופה לכך, ציידנו אתכם בדוגמא לשימוש ב- vl_vector.
- זמן ההגשה (שימו לב לבונוס!):
  - **תאריך הגשה: יום חמישי 13/01/2022 בשעה 23:55.**
  - **הגשה באיחור**: ניתן להגיש את התרגיל באיחור של עד 24 שעות, **בקנס** של 10 נקודות.
  - **הגשה מוקדמת**: על כל יום (24 שעות) שתגישו מוקדם יותר, תקבלו 2 נקודות **בונוס**. עד חמישה ימים וסה"כ 10 נקודות (מי שיגיש יותר מוקדם מ-5 ימים יקבל 10 נקודות).

זמן הגשה	עד ה-08/01 ב- 23:55	עד ה-09/01 ב-23:55	עד ה-10/01 ב-23:55	עד ה-11/01 ב-23:55	עד ה-12/01 ב-23:55	עד ה-13/01 ב-23:55	עד ה-14/01 ב- 23:55
בונוס \ קנס	+10	+8	+6	+4	+2	0	-10

5 חומרי עזר

5.1 תוכנית לדוגמא - Highest Student Grade

יצרנו עבורכם תוכנית לדוגמא המשתמשת בכמה מהתכונות הבסיסיות של הווקטור. תוכלו לקמפל ולהריץ את התוכנית. התוכנית highest_student_grade.ccp נמצאת בתיקיית Example במודל (ואין להגישה).

התוכנית קולטת רשימה של סטודנטים מהמשתמש דרך ה-CLI, ומדפיסה את הסטודנט עם הממוצע הגבוה ביותר. התוכנית מגדירה מחלקה בשם Student, שלה 2 שדות "שם פרטי" ו-"מוצע ציונים". התוכנית עושה שימוש ב-vl_vector לשמירת הסטודנטים שנקלטו על ידי המשתמש.

דוגמת הרצה:

```
$ ./HighestStudentGrade
Enter a student in the format "<name> <average>" or an empty string to stop:
Mozart 70.5
Enter a student in the format "<name> <average>" or an empty string to stop:
Beethoven 95
Enter a student in the format "<name> <average>" or an empty string to stop:
Liszt 83.0
Enter a student in the format "<name> <average>" or an empty string to stop:
<< Note: This is an empty line >>
-----------------------------------------------------------------
Total Students: 3
Student with highest grade: Beethoven (average: 95)
```

הקלט שהזין המשתמש צבוע בירוק והשורה לפני שורת הפסים ריקה כי המשתמש הזין קלט ריק. נדגיש:

- התוכנית מבצעת בדיקות קלט בסיסיות בלבד. תוכנית זו אינה מתיימרת להיות פתרון מלא ומקיף, אלא להציג שימוש בסיסי ב-vl_vector שיצרתם.
- אנו ממליצים כי תעיינו בקפידה בתוכנית, הכוללת הערות המסבירות את הנעשה שלב שלב. תוכנית זו תוכל לסייע לכם בהבנת המשימה.

5.2 קבצי ה-Pre-submission

קוד המקור של תוכנית ה-Presubmit זמינה עבורכם במחשבי האקווריום, ותוכלו למצוא שם בדיקות בסיסיות של הווקטור, **לרבות בדיקת Resize בסיסית**. מומלץ בחום שתעיינו בתוכנית זו. אתם רשאים לבצע שינויים בתוכנית זו בכדי ליצור Tests משלכם.

5.3 דוגמא לגדילת וכיווץ הווקטור

בטבלה הבאה תיאור מקרה אחד של הגדלה וכיווץ. שימו לב כי חלק מהפעולות מתוארות במילים וחלק כקוד. כמו כן, ערכי ה-size וה-capacity הם הערכים שמתקבלים **כתוצאה** מביצוע הפעולה.

הסבר	גודל size	קיבולת capacity	פעולה
ערכי ברירת מחדל	0	16	vl_vector <int> vec;
	1	16	vec.PushBack(1);

Insert 16 additional items, **one by one**	25	17	נוסיף את האיברים אחד אחר השני עד שנגיע לאיבר ה-16. שם נדרש מעבר לזיכרון דינמי (כי 16+1>C) וחישוב הקיבולת: $cap_{16}(16,1) = \left\lfloor \frac{3 \cdot (16+1)}{2} \right\rfloor = 25$
Insert 13 additional items, **using an iterator** (at one single call to "insert")	45	30	כמות האיברים שנרצה לשמור בווקטור (30) חוצה את הקיבולת, לכן נחשב לפי cap_C: $cap_{16}(17,13) = \left\lfloor \frac{3 \cdot (17+13)}{2} \right\rfloor = 45$
Erase 13 items, **one by one**	45	17	כשמסירים איברים (אך לא "חוזרים" לזיכרון הסטטי) ה-capacity לא קטן.
vec.Clear();	16	0	הקיבולת מאותחלת חזרה ל-C.
Insert 17 items, **one by one**	25	17	ה-capacity לא יהיה 45 כיוון שבשלב הקודם חזרנו להשתמש בזיכרון הסטטי, פעולה ש-"אתחלה" את ה-capacity הדינמי.