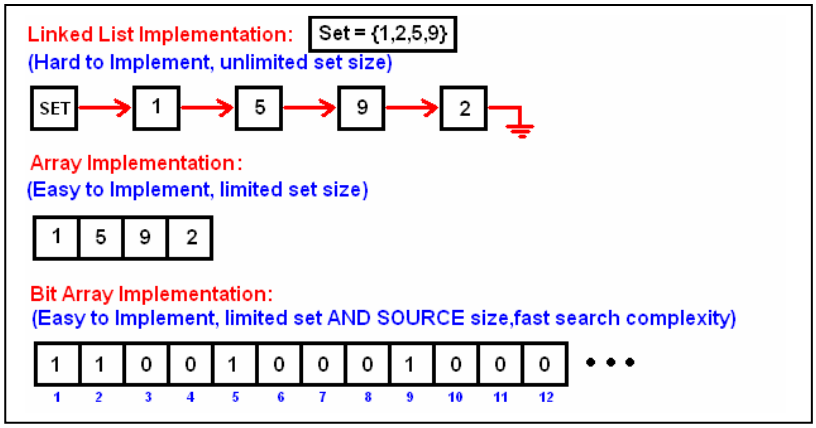
# ADTS

## Set

- מומלץ להשתמש בשביל לממש אוסף של איברים שאין בהכרח חשיבות לסדר ביניהם או קשר כלשהו אחר ביניהם.

### תיאור סכמטי:




### טיפוסי נתונים:

Set מצביע ל Set_rec שמכיל רשימה של אלמנטים, פונקציות לפעולות על אלמנטים (העתקה, השוואה, שחרור, הפיכה למחרוזת) בד"כ איטרטור, מצביע לאיבר הנוכחי ולאיבר האחרון.
Element טיפוס איבר גנרי מסוג *void
Result טיפוס enum של {Failure,Success}

### פקודות מאקרו:

**SET_FOREACH(e,s)** לולאת מעבר על כל אלמנט e בקבוצה s וקידום האיטרטור הפנימי בקבוצה **לכן אסור קינון של לולאות עבור אותה קבוצה!! ואסור שימוש של המאקרו בתוך לולאה המשתמשת גם היא באיטרטור של S**
**SetIsEmpty(s)** האם הקבוצה S ריקה

### פונקציות:

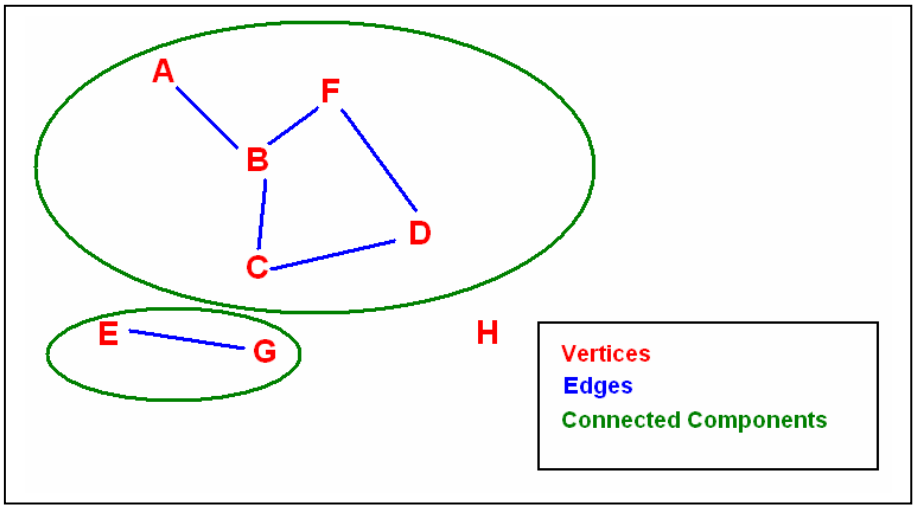
| שם | מה מוחזר | מה הפונ' עושה |
|---|---|---|
| Set **SetCreate**(bool cmp(Element,Element), Element Cpy(Element), Lbl(Element),Fre(Element)) | ה**מצביע** לגרף החדש (או **NULL** במקרה של כשלון) | יוצרת את הקבוצה ע"י שימוש במצביעים לפונקציות ההעתקה, ההשוואה, השחרור וההפיכה למחרוזת (כדי לדעת איך לעבוד עם אלמנטים). |
| void **SetDestroy**() | - | הורסת את הקבוצה. |
| Result **SetAdd** (Set, Element) | **SUCCESS** במקרה של הצלחה **וגם אם האיבר נמצא כבר**<br>**FAILURE** עבור כל מקרה אחר | מוסיפה קודקוד לGRAPH. |
| Result **SetRemove**(Set, Element) | **FAILURE** במקרה שהאיבר לא נמצא בGRAPH.<br>**SUCCESS** במקרה של הצלחה | מסירה איבר מהקבוצה |
| bool **SetIsIn**(Set, Element) | **TRUE** אם נמצא **FALSE** אם לא | האם האיבר נמצא בקבוצה? |
| int **SetSize**(Set) | **מספר האיברים** בקבוצה | מהו גודל הקבוצה |
| int **SetMaxSize**(Set) | **גודל מירבי** (**0** או **-1** אם אין הגבלה) | מהו הגודל המירבי של הקבוצה |
| Set **SetIntersection**(Set,Set) | **קבוצה חדשה** ובה האיברים המשותפים לשתי הקבוצות. **NULL** אם אין חיתוך או אם נכשלה הבנייה של קבוצת החיתוך | קבוצת חיתוך בין שתי קבוצות * |
| Set **SetUnion**(Set,Set) | **קבוצה חדשה** ובה איחוד כל האיברים משתי הקבוצות. **NULL** אם אין חיתוך או אם נכשלה הבנייה של קבוצת החיתוך | קבוצת איחוד בין שתי קבוצות * |
| Void **SetPrint**(Set) | - | הדפסת הקבוצה |
| Element **SetFirst**(Set) | **עותק של האיבר הראשון** בקבוצה.,**תופעת לוואי:** איפוס האיטרטור הפנימי | האיבר הראשון בקבוצה |
| Element **SetNext**(Set) | **עותק של האיבר הבא בקבוצה**, NULL אם האיבר הנוכחי הוא האחרון.**תופעת לוואי:** קידום האיטרטור הפנימי | האיבר הבא בקבוצה |

*** אנו יוצאים מנקודת הנחה שלשתי הקבוצות אותם אלמנטים ואותן פונקציות טיפול עבור כל אלמנט**

## Graph

- מומלץ להשתמש בשביל לממש קבוצה שיש בה יחס או קשר בין שני איברים כלשהם.
- גרף הוא NestedADT כלומר משתמש בSet ADT (מימוש פרטי שכל אלמנט הוא מחרוזת)

תיאור סכמטי:




טיפוסי נתונים:

Graph מצביע ל Graph_rec שמכיל רשימה של קודקודים (VERITCES) וקשתות (EDGES)

Label מחרוזת שמייצגת שם קודקוד

Result טיפוס enum של {Failure,Success}

פונקציות:

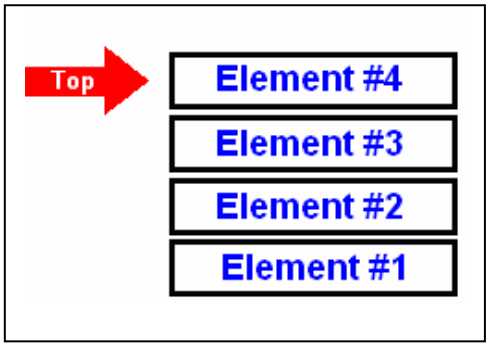
| מה הפונ' עושה | מה מוחזר | שם |
|---|---|---|
| יוצרת את הגרף. * | ה**מצביע** לגרף החדש (או NULL במקרה של כשלון) | Graph **GraphCreate**() |
| הורסת את הגרף. | - | void **GraphDestroy**() |
| מוסיפה קודקוד לGRAPH. | **SUCCESS** במקרה של הצלחה **וגם אם האיבר נמצא כבר** **FAILURE** עבור כל מקרה אחר | Result **GraphAddVertex**(Graph, Label) |
| מוסיפה קשת לגרף משני שמות קודקודים. | **FAILURE** במקרה שאחד מהקודקודים לא נמצא בGRAPH או שההוספה נכשלה. **SUCCESS** במקרה של הצלחה | Result **GraphAddEdge**(Graph, Label, Label) |
| מסירה קשת | **FAILURE** במקרה שהקשת לא נמצאה בGRAPH. **SUCCESS** במקרה של הצלחה | Result **GraphRemoveEdge**(Graph, Label, Label) |
| מסירה קודקוד | **FAILURE** במקרה שהקודקוד לא נמצא בGRAPH. **SUCCESS** במקרה של הצלחה | Result **GraphRemoveVertex**(Graph, Label) |
| בניית קבוצת השכנים של הקודקוד בגרף | **קבוצת קודקודים שכנים** לקודקוד (טיפוס SET) **NULL** במקרה שקרה כשלון בבניית קבוצת השכנים. | Set **GraphNeighbours**(Graph, Label) |
| בניית רכיב הקשירות של הקודקוד בגרף** | **הקבוצה המייצגת את רכיב הקשירות** של הקודקוד. **NULL** במקרה של בעייה בבנייה שלה | Set **GraphGraphConnectedComponents**(Graph, Label) |
| מדפיסה את הגרף | - | Void **GraphPrint**(Graph) |

* שלב יצירת הגרף ממומש ע"י בניית SET שכל איבר בו הוא מחרוזת (לכן נשלחות פונקציות לטיפול במחרוזות)

** שימוש רקורסיבי בתוצאה ובקבוצה שמועברת כפרמטר לפונקציה DEPTH FIRST TRAVERSAL שממומשת באופן רקורסיבי מציאת שכן לכל שכן (ובדיקה שלא הוספנו אותו כבר לקבוצה)

## Stack

- מומלץ להשתמש בשביל לממש קבוצה של יחסי LAST IN FIRST OUT.




תיאור סכמטי:

טיפוסי נתונים:
Stack מצביע ל Stack_t שמכיל אוסף של איברים
Elem טיפוס איבר גנרי מסוג *void
Cpy_Func פונקצית העתקה של איברים
Free_Func פונקצית שחרור של איברים
Result טיפוס enum של {Fail,Success}

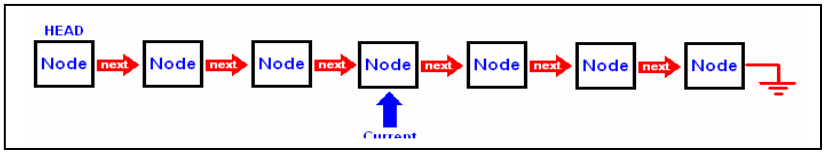
פונקציות:

| שם | מה מוחזר | מה הפונ' עושה |
|---|---|---|
| Stack **create**(int max_size, cpy_func, free_func) | **מחסנית חדשה** בגודל מקסימלי של MaxSize | יוצרת את המחסנית. |
| Void **destroy**(Stack) | - | הורסת את המחסנית |
| Result **push** (Stack, Element) | **FAIL** אם האיבר או מצביע המחסנית הם NULL.<br>**SUCCESS** במקרה של הצלחה | דחיפת איבר לראש המחסנית |
| Result **pop** (Stack) | **FAIL** אם המחסנית NULL<br>**SUCCESS**במקרה של הצלחה | הסרת האיבר בראש המחסנית |
| Result **top** (Stack, Element*) | עותק של האיבר בראש במחסנית.<br>**FAIL** אם האיבר או מצביע המחסנית הם NULL.<br>**SUCCESS** במקרה של הצלחה | האיבר הראשון במחסנית |
| int **count** (Stack) | **מספר האיברים** במחסנית.<br>**(-1)** אם המצביע למחסנית הוא NULL. | מספר האיברים במחסנית |

## LinkedList (מתוך מת"מ 2)

- מומלץ להשתמש כדי לממש רשימת איברים מקושרת עם אופציה של סינון ומיון איברים

תיאור סכמטי:




טיפוסי נתונים:
ListElement טיפוס איבר גנרי מסוג *void
LinkedList מצביע לSTRUCT LinkedList_t
FilterElement טיפוס איבר הסינון (עבור פונקצית הסינון) מסוג *void - אפשרי להשתמש בו בתור הערך שלפיו תסונן הרשימה, ניתן להעביר אותו לרשימה בצירוף מצביע לפונקציית הסינון כדי שתשתמש בו. הפונקציה החדשה תחזיר TRUE רק עבור כל איבר FilterElement
copyElemFunc מצביע לפוקנצית ההעתקה שמקבלת ומחזירה ListElement (ניתן להחזיר עותק של האיבר המועבר או להחזיר מצביע אליו)
freeElemFunc מצביע לפוקנצית השחרור שמקבלת ListElement ומשחררת אותו (ניתן להעביר פונקציה שלא עושה כלום ואז לא ישוחררו האיברים)
filterElemFunc מצביע לפוקנצית הסינון שמקבלת ListElement וFilterElement ומחזירה 1 אם האיבר תואם את הסינון או 0 אם האיבר לא תואם את הסינון (ניתן להשתמש בה כפונקצית העתקה שפשוט תחזיר 1 תמיד)
cmpElemFunc מצביע לפוקנצית ההשוואה (השוואה למטרות מיון) שמקבלת שני ListElement ומחזירה (-1) אם האיבר הראשון צריך להיות לפני האיבר השני, 1 אם האיבר השני צריך להיות לפני האיבר הראשון ו0 אם הם שווים
equalElemFunc מצביע לפוקנצית ההשוואה (השוואה רגילה) שמקבלת שני ListElement ומחזירה 1 אם האיברים זהים ו0 אם הם שווים
ListResult טיפוס enum של
{LIST_SUCCESS, LIST_FAIL ,LIST_BAD_ARGUMENTS, LIST_OUT_OF_MEMORY}

פונקציות:

| שם | מה מוחזר | מה הפונ' עושה |
|---|---|---|
| LinkedList **LinkedList_create**(copyElemFunc, freeElemFunc) | **מצביע** לרשימה החדשה (או NULL במקרה של כשלון). | יוצרת רשימה חדשה ומשייכת לכל האיברים פונקציות העתקה ושחרור |
| LinkedList **LinkedList_filterElements**(LinkedList, copyElemFunc, freeElemFunc, filterElemFunc, FilterElement) | **רשימה חדשה** ובה האיברים שעברו את הסינון (אזהרה: אם פונקצית ההעתקה לא תחזיר עותק אזי ברגע שנשחרר איבר מהרשימה החדשה המצביע ברשימה המקורית יצביע על איזור משוחרר) | מקבלת רשימה מקורית, פונקצית העתקה + פונקצית שחרור (שמועברות לרשימה החדשה) פונקצית הסינון ואיבר הסינון |
| Unsigned long **LinkedList_getNumElements**(LinkedList) | **מספר האיברים** ברשימה | מחזירה את מספר האיברים ברשימה |
| LinkedResult **LinkedList_sortElements**(list cmpElemFunc) | **LIST_SUCCESS** במקרה של הצלחה. (והרשימה שמועברת כפרמטר ממוינת)<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL. | ממיינת את הרשימה |
| ListResult **LinkedList_destroy**(LinkedList) | **LIST_SUCCESS** במקרה של הצלחה.<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL | הורסת את הרשימה |
| ListResult **LinkedList_insertFirst**(LinkedList,ListElement) | **LIST_SUCCESS** במקרה של הצלחה.<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_OUT_OF_MEMORY** כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL | הכנסת איבר לראש הרשימה |
| ListResult **LinkedList_insertBeforeCurrent**(LinkedList, ListElement) | **LIST_SUCCESS** במקרה של הצלחה.<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_OUT_OF_MEMORY** כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL | הכנסת איבר לפני האיבר הנוכחי |
| ListResult **LinkedList_insertAfterCurrent**(LinkedList, ListElement) | **LIST_SUCCESS** במקרה של הצלחה.<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_OUT_OF_MEMORY** כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL | הכנסת איבר אחרי האיבר הנוכחי |
| ListResult **LinkedList_getCurrent**(LinkedList, ListElement*) | **LIST_SUCCESS** במקרה של הצלחה. **בנוסף בפרמטר יוחזר המצביע לאיבר הנוכחי.**<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_FAIL** הרשימה שהועברה היא ריקה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL. | החזר את האיבר הנוכחי המוצבע במסחנית |
| ListResult **LinkedList_removeCurrent**(LinkedList) | **LIST_SUCCESS** במקרה של הצלחה.<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_FAIL** הרשימה שהועברה היא ריקה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL. | הסר את האיבר הנוכחי המוצבע במסחנית |
| ListResult **LinkedList_goToHead**(LinkedList) | **LIST_SUCCESS** במקרה של | לך לאיבר בראש הרשימה |

| | | |
|---|---|---|
| | הצלחה. **תוצר לוואי: האיטרטור יצביע לאיבר הראשון ברשימה**<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL | |
| לך לאיבר הבא ברשימה | **LIST_SUCCESS** במקרה של הצלחה. **תוצר לוואי: האיטרטור יצביע לאיבר הבא ברשימה**<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_FAIL** במקרה שהרשימה ריקה או שהאיבר הנוכחי היה האחרון | ListResult **LinkedList_goToNext**(LinkedList) |
| מצא האיבר והצבע עליו | **LIST_SUCCESS** במקרה של הצלחה. **תוצר לוואי: האיטרטור יצביע למופע הראשון ברשימה של האיבר שהועבר**<br>**LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL<br>**LIST_FAIL** במקרה שהרשימה ריקה או שהאיבר לא נמצא ברשימה | ListResult **LinkedList_find**(LinkedList, ListElement, equalElemFunc) |

# C-SHELL

**תחילת הסקריפט (קריאה לTCSH מבלי לקרוא את קבצי האתחול שנקראים בד"כ):**

```
#!/usr/local/bin/tcsh –f
```

## עבודה עם קבצים ומדריכים

**קבלת השם המלא של המדריך הנוכחי:**

```
pwd
```

**הדפסת הקבצים במדריך הנוכחי:**

```
ls
```

**הדפסת הקבצים בתת המדריך subdir:**

```
ls subdir
```

**הדפסת הקבצים במדריך הנוכחי עם יתר פירוט:**

```
ls –l
-rw-r--r--    1    adi    5366        5434    Oct 19 21:40    adi.txt
drwxr-xr-x    5    adi    5366        1024    Oct 19 21:40    subdir
```

| Permissions | directories | owner | group | size | date | File |
|---|---|---|---|---|---|---|
| -rw-r--r-- | 1 | adi | 5366 | 5434 | Oct 19 21:40 | adi.txt |
| drwxr-xr-x | 5 | adi | 5366 | 1024 | Oct 19 21:40 | Subdir |

**בכדי לבצע מעבר על כל הקבצים:**

```
foreach F(*)
        echo $F                ← הדפסת שם הקובץ
end
```

**בכדי לבצע מעבר על כל הקבצים בסיומת C:**

```
foreach F(*.c)
        echo "file name" $F            ← הדפסת שם הקובץ
end
```

**בכדי לבדוק האם המשתנה F הוא מדריך (קובץ):**

```
if (-d (–f) $F) then
…
endif
```

**מחיקת קובץ**

```
rm
```

**כתיבה לקובץ (אם הוא לא קיים יוצרים אותו)**

```
command  !>> file
```

## עבודה עם תכנים

**קרא שורה שורה מהקלט (אין אפשרות לקרוא מקובץ אלא רק מהSTANDARD INPUT)**

```
set line = ($<)
while (${#line}!=0)
… set line = ($<)   ← קידום לשורה הבאה
end
```

**הדפסת תוכן קבצים (ומספור שורות) לפלט הסטנדרטי:**

```
cat (-n) file1 file2 file3
```

**הדפסת n שורות ראשונות של קבצים לפלט הסטנדרטי:**

```
head -n file1 file2 file3
```

**הדפסת n שורות אחרונות של קבצים לפלט הסטנדרטי:**

```
tail -n file1 file2 file3
```

**הדפסת השורות אחרונות של קבצים לפלט הסטנדרטי החל מהשורה ה:**

```
tail +n file1 file2 file3
```

**מיון שורות קבצים:**

```
Sort [options] (files)
Options:
```

| | |
|---|---|
| -b | התעלם מרווחים בתחילת שורה |
| -d | סדר בסדר מילוני |
| -f | התעלם מהבדלי גודל אותיות |
| -m | מזג קבצים ממויינים לקובץ אחד |
| -n | מיין לפי ערך מספרי מהקטן לגדול |
| -ofile | הדפס פלט לקובץ file |
| -r | הפוך את סדר המיון |
| -u | סנן שורות זהות |
| +n | דלג על n המילים הראשונות |

**הדפס את השורות בקובץ שמכילות את המחרוזת string**

```
grep [options] string
Options:
```

| | |
|---|---|
| -c | הדפס רק את סך כל מספר השורות |
| -h | הדפס את השורות עצמן ללא שמות הקבצים בהם הן נמצאות |
| -i | התעלם מהבדלי גודל אותיות |
| -l | הדפס את שמות הקבצים ללא השורות |

| | |
|---|---|
| -n | הדפס את השורות ואת מספרן בקבצים |
| -v | הדפס את השורות בהן **לא מופיעה** המחרוזת |
| -w | הדפס את השורות בהן מופיעה המחרוזת **בדיוק ולא כתת מחרוזת** |
| ^string | הדפס את השורות שמתחילות במחרוזת |

**הפרד בין מילים מתוך הקלט ע"י שימוש בתו המפריד c**

```
cut –d"c"
```

**בחר רק את המילה הראשונה (התו הראשון) בכל שורה**

```
cut –d" " –f1 (–c1)
```

**בחר רק את ארבעת התוים הראשונים בכל שורה**

```
cut –d" " –c1-4
```

**ספור מילים/תווים/שורות**

```
wc [options]
```

Options:

| | |
|---|---|
| -c | הדפס רק את מספר התווים |
| -l | הדפס רק את מספר השורות |
| -w | הדפס רק את מספר המילים |

*** בסוף גם מודפס שם הקובץ**

**הדפסה מפורמטת לפי תווי פורמט**

```
printf "[strings]" "[ and format strings]" [more strings]" $var1
```

format strings:

| | |
|---|---|
| %s | הדפס מחרוזת |
| %X.Ys | הדפס מחרוזת שתתפוס מקום של X תווים – הדפס רק Y תווים מתוך המחרוזת |

## עבודה עם משתנים ארגומנטים ורשימות

**הגדר את המשתנה X ושים לתוכו את הערך Y**

```
set X = Y
```

Y יכול להיות: מספר - 123
מחרוזת – abc
רשימה של מחרוזות ומספרים – (abc 555 123)
מחרוזת של יותר ממילה אחת – "abc def ghi"

**ערך המשתנה VAR (GET)**

```
$VAR
```

**בצע פעולה חשבונית על VAR (SET)**

```
@VAR = 4; @VAR++; @VAR = $ VAR + 5l
```

**צור רשימה**

```
set mylist = (123 456 789)
```

**ערך האיבר ה-n ברשימה (לשים לב שבניגוד לC אינדקס האיבר הראשון הוא 1, לא 0)**

```
$mylist[n]
```

**מספר האיברים ברשימה**

```
${#mylist}
```

**תת הרשימה מהאיבר ה-m עד לאיבר ה-n**

```
Mylist[m-n]
```

**תת הרשימה מהאיבר הראשון עד לאיבר ה-n**

```
Mylist[-n]
```

**תת הרשימה מהאיבר ה-m עד לאיבר האחרון**

```
Mylist[m]
```

**קבוצה המרוכבת מכל איברי הרשימה**

```
Mylist[*]
```

**ערך הארגומנט הראשון שמועבר לסקריפט (0 זו הפקודה עצמה)**

```
${1} או ${argv[1]}
```

**רשימת כל הארגומנטים**

```
$*
```

**מספר הארגומנטים המועברים לסרקיפט**

```
${#argv}
```

**רשימה פחות איבר**

```
${list}/$f
```

## עבודה עם פקודות

**קינון פקודות ע"י שימוש ב BACKQUOTES ` `**

```
echo you have `ls | -wc –l` files in `pwd`
```

**העברת קלט בין פקודות (PIPELINING)- פקודה מס' 2 תיקח את כקלט את הפלט של פקודה מס' 1:**

```
(Command 1) | (Command 2)
```

**תחביר פקודות**

```
`echo this is a command`
"the value for VAR is $VAR"
'this string will be printed as-is'
```

## עבודה עם מבני בקרה

**חזור על הפקודה n פעמים**

```
repeat n (command)
```

**חזור על בלוק בתנאי**

```
while  (condition)
....
end
```

**בצע עבור כל איבר ברשימה**

```
foreach F (mylist)
        (do-something with F, $F etc.)
end
```

**בצע בתנאי**

```
 If (condition) then
        (do-something)
else
        (do-something-else)
endif
```

**אופרטורי השוואה חשבוניים** <,>,>=,=<

**אופרטורי השוואה למחרוזות** ==,=~,~!,=!

**~= המחרוזת היא מהצורה**
**~! המחרוזת לא מהצורה**
**== המחרוזת היא שווה ממש ל**
**=! המחרוזת לא שווה ממש ל**
Str?ng **השלם את תו אחד במקום**
Str* **השלם את כל התווים אחרי במקום**
*Str **השלם את כל התווים לפני**

**אוסף תנאים**

```
switch ($varString)
        case "a":
                (do something)
        breaksw
        case "b:
                (do something else)
        breaksw
        default:
                (do another thing)
        breaksw
endsw
```

**עבור ל**

```
goto label
…
label: (do something)
```

# C++

## Reference, Const Reference

**הגדרת משתנה מתייחס (Reference) למשתנה אחר (המשתנה מתחפש למשתנה המקור)**

```
int i;
int j = &i;
```

- מרגע ההגדרה כל השמה למשתנה אחד תשנה את ערכי שני המשתנים
- חובה להגדיר משתנה מתייחס באיתחולו (לא ניתן לבצע השמת התייחסות מאוחר יותר)
- לא ניתן לשנות התייחסות – עד לסוף הבלוק בו חי המשתנה המתייחס הוא יתייחס לאותו משתנה
- **לא לבלבל** עם הביטוי & Cm שמשמעותו כתובת!

**שימוש עיקרי: העברת פרמטר By Reference לפונקציה**

```
void Swap (int &i, int &j)
{
        int t = j;
        j = i;
        i = t;
}
Swap(a,6); ← Const reference שגיאת קומפילציה, יעבוד רק עם
```

**החזרת Reference למשתנה**

```
int& index (int a[], int indx)
{
        Return a[index];
}
int a = index(a,7);
a = 6 ← כעת שונה גם האיבר במערך
```

**Const Reference(עצם זמני שהפונצקיה מתחייבת לא לשנות)**

```
double sqr (const double& a)
{
        a = 6.0 ← שגיאת קומפילציה
}
```

## Function Overloading

**הגדרת מספר פונקציות בעלות אותו שם (פונקציה מזוהה ע"י השם שלה והפרמטרים שלה)**

```
void Swap (int &i, int &j)
void Swap (char &i, char &j)
```

**חוקים להמרה**

1. המרה מדוייקת + טריביאלית (int->int&, int ->const int)
2. המרה שכוללת הרחבת המשתנה (char,short int->int,int->double)
3. המרה רגילה (int->float,float->int)
4. המרה רגילה שמוגדרת ע"י המשתמש

## Namespaces

- הרחבה לבלוקים של קוד, ניתן יהיה להגדיר ולהשתמש ב namespace אחד בכמה קבצים
- אם משתמשים בnamespace עדיף להצהיר על פונקציה ספציפית

```
Using namespace1::func1
```

כדי להימנע מהתנגשות בין שמות ולהימנע מניפוח של הקובץ בגלל טעינת רכיבים מיותרים

## I/O

```
#include <iostream>
using std::cin;
using std::cout;
int i;
cin >> i;
cout << "bla bla" << i;
```

## Classes

אוסף של משתנים ומתודות – בניגוד לstruct, במחלקות משתנה מוגדר כברירת מחדל כprivate

**מצייני גישה**

```
class C {
```

```
        private:
        int a=0;
        public:
        int b=0;
        protected:
        int c=0;
}
```

השימוש בa יוכל להתבצע רק ע"י member של הC class (או friend), השימוש בb יוכל להתבצע ע"י כל קטע קוד בתוכנית, השימוש בc יוכל להתבצע רק ע"י member של הC class או של מחלקות היורשות ממנה

### this

this היא דרך הפנייה לעצם עצמו בזמן ריצה – בניגוד לclass שהוא למעשה רק תבנית של הכנת עצם

### Const, static and friends

| שם | מאפיינים | דוגמא |
|---|---|---|
| **פונקציית CONST** | לא משנה את הTHIS<br>עצמים שמוגדרים כconst<br>יכולים להפעיל רק את<br>הפונקציות האלה כי הן<br>מתחייבות שאינן משנות<br>אותם. | Class String<br>{<br>public:<br>int length() const;<br>} |
| **פונקציית STATIC** | אין לה THIS ניתן לקרוא<br>לה ישירות מתוך הSCOPE<br>של הCLASS. פונקציה כזו<br>היא למעשה שרות שניתן<br>לחשוב על השימוש בו<br>כפונקציה רגילה בC | Class String<br>{<br>public:<br>static char* ToLower(char* s)<br>{…}<br>}<br>char* s1 = String::ToLower("ABC"); |
| **פונקציות או מחלקות FRIEND**<br>**לא משתמשים בFRIEND במבחן!**<br>**כלMETHOD FRIEND ניתן להחליף בMETHOD גלובלי שקורא לMETHOD PUBLIC שניגשת לשדות PRIVATE** | פונקציות non member או<br>מחלקות שמחלקה נותנת<br>להן גישה למשתני<br>PRIVATE שלה | Class String<br>{<br>friend int func1();<br>private :<br>int a;<br>}<br>int func1()<br>{<br>String s;<br>s.a = 123;<br>} |

## Templates (בניית תבנית קוד גנרית)

### Function templates

ראינו עבור פונקציה שמבצעת את אותו תהליך עבור מספר שונה של פרמטרים או סוגים שונים של פרמטרים ניתן היה להגדיר OVERLOADING לפונקציה (כלומר לכתוב אותה ספציפית עבור כל משתנה) רמה גבוהה יותר ניתן להשיג במידה ויש מספר קבוע של משתנים מטיפוס מסויים ואנחנו רוצים לחזור על אותה פעולה עבור כל טיפוס באופן הבא:

```
template<class T>
void Swap(T& i, T& j) {
        T t = j;
        j = i;
        i = t;
}
```

למעשה הקומפיילר ייצור "חורים" שאותם הוא יימלא בזמן ריצה כאשר הפונקציה Swap תיקרא

**חובה ששני המשתנים שמועברים יהיו מאותו סוג – הקוד יכשל אפילו אם ניתן להמיר את אחד סוגים לסוג המשתנה השני כיוון שלא מאופשר השילוב (המסוכן) של המרות ותבניות.**

### חוקים לעדיפות הקריאה (עם תבנית)

1. הפונקציה המפורשת עבור המשתנים (אם כתובה כזו).
2. הפונקציה הגנרית עם התבנית.
3. פונקציה מפורשת עם המרות של אחד המשתנים.

### Class templates

בדומה לפונקציה גנרית ניתן להגדיר מחלקה גנרית לדוגמא מחסנית גנרית (לשים לב שכל פונקציה ממומשת כ TEMPLATE FUNCTION **כולל C'TOR**):

| Stack.h | Stack.cc |
|---|---|
| Template <class T><br>class stack {<br>int top_index,size;<br>T* array;<br>}<br>public:<br>stack (int s = 100);<br>void pop();<br>void push(T e);<br>T top();<br>int size;<br><br>Template <class T> ← template function for c'tor<br>Stack<T>::stack(int s) {<br>top_index = 0; size = s;<br>array = new T[s];<br>}<br>….. | Stack<int> stack_i(100)<br>Stack<char*> stack_c(5)<br>… |

## Operator Overloading

```
Class Complex
{
…
        private: double real, double im;
        Complex operator + (complex c1)
        { return complex(real + c1.real, im + c1.im) }
}
```

**הגדרות אופרטורים**

| | |
|---|---|
| Complex operator + (complex c1) | יכול לשנות הכל |
| Complex operator + (const complex& c1) | לא משנה את הארגומנט הימני |
| Complex operator + (complex c1) const | לא משנה את הארגומנט השמאלי (שהוא למעשה ה THIS או ה HIDDEN ARGUMENT |

### אופרטור הדפסה והמחלקות istream ostream

| ISTREAM | מה הפונקציה עושה |
|---|---|
| int i = cin.get(); | קבל תווים מהקלט |
| int i = cin.peek(); | קבל תווים מהקלט **בלי לקדם את סמן הקלט (כך ניתן לקרוא את אותו תו כמה פעמים)** |
| char *s = cin.peek(); | קרא שורה שלמה מהקלט (לא עוצר ברווחים) |
| **OSTREAM** | |

| | |
|---|---|
| cout.pecision(4) | דיוק של 4 ספרות אחרי הנקודה |
| cout.width(8) | גודל BUFFER של 8 תווים סה"כ |
| endl | תו סוף שורה |
| ends | תו סוף מחרוזת |

```
cout << 12343 << endl;
```

**הוספת אופרטור הדפסה לCLASS (גירסאת FRIEND)**

**לא משתמשים בFRIEND במבחן!**

```
class Complex
{
        private: double real, im;
        public:
        friend istream& operator >> (istream & in, Complex& Z);
        friend ostream& operator << (ostream & out, const Complex& Z);

}
```

**הוספת אופרטור הדפסה לCLASS (גירסאת NON-FRIEND)**

```
class Complex
{
        private: double real, im;
        public:
        ostream& print (ostream& os) const; ← חסרון: הוספת פונקציה PUBLIC
        { return os << real << "+ " << im << "i"; }
        ostream& operator << (ostream & out, const Complex& Z);

}
ostream& operator << (ostream & out, const Complex& Z)
{return Z.print(out)};
```

## Ct'or, D'tor, Copy C'tor and operator =

על פי ה"רוח של ++C" מצב בו עצם מאותחל בלי ערך מוגדר הוא מצב **לא תקין** לכן לכל מחלקה ישנה פונקציה שהיא בעלת שם המחלקה שנקראת CONSTRUCTOR שמטרתה שכשאר מוגדר עצם למחלקה היא תאתחל את המחלקה וכל ה DATA MEMBERS שלה. באם לא הוגדר CTOR הקומפיילר מקצה אוטומטית CTOR חסר ארגומנטים שלא עושה כלום (או לפחות לא משהו שבריא להסתמך עליו...) בנוסף חייבים להגדיר פונקציה שנקראת DESTRUCTOR (במיוחד אם מקצים זכרון באופן דינמי) ברוב המקרים לא קוראים לה היא נקראת אוטומטית כשה SCOPE של העצם מת (הבלוק בו הוא מוגדר נגמר או אם נזרק EXCEPTION שלא נתפס על ידיו)

```
Class Stack
{
        Public:
        Stack (int size): (רשימת אתחול){}; ←The C'tor MUST be public an it is NOT inherited
        ~Stack ();←same is the D'tor and in addition the D'tor MUST NOT contain any arguments (so it will be called automatically)
}
```

**כללים (חלקם כללי אצבע):**

- ניתן לשים ערכי DERAULT לCTOR כדי לאפשר לCOMPLIER חופש לקריאה גם מידה ולא הועברו כל הארגומנטים.
  Stack (int size **= 0**): (רשימת אתחול){};
- מרגע שמומש **אזשהו** CTOR, הCTOR חסר הארגומנטים לא קיים יותר.
- CTOR טוב הוא CTOR ריק מאחר וכל הMEMBERS של הCLASS נוצרים בשלב רשימת האתחול (מלבד משתנים סטטיים) – לכן מאתחלים אותם בשלב רשימת האתחול. לדוגמא:

  Stack (int size): <= לכן כאן תהיה רשימת האתחול data members כאן למעשה נוצרים כל ה =>{};
- **אסור לאתחל משתנים סטטיים ברשימת האתחול**
- ברשימת אתחול של מחלקה יורשת (Derived) **חייב להופיע CTOR של המחלקה הנורשת (Base)**
- **CTOR** אף פעם לא יכול להיות VIRTUAL, DTOR יכול להיות!
- במידת האפשר, הDTOR לא יכול לזרוק EXCEPTIONS כי אין מה לעשות איתם.

במקרים בהם נדרש **אופרטור השמה** לCLASS ניתן לבצע העמסה שלו. דוגמא

```
Class Numbers {
private: int* array_holder;
         int size;
public:
                Person (int s=0): array_holder = new int[s]; size =s; {}….
}
…
Numbers c1 (12), c2 (15);
…
c1=c2;
```

```
…
Numbers::operator= (const Numbers & right)
{
        If (this == &right) return *this
/* נמנע מהשמה עצמית כדי למנוע מצב שאחרי שחרור ננסה לשים ערך חדש על איזור ששוחרר וכבר לא שייך לתוכנית */
        Delete [] array_holder;
        array_holder = new array [right.size]
        (COPY ARRAYS…)
        return *this;
}
```

אם היינו מבצעים את השורה הבאה היה נקרא **COPY CONSTRUCTOR**:

```
Numbers c1 (12), c2 (c1);
Numbers (const Numbers& source) {…}
```

**כללי אצבע:**

- מומלץ לשתף קוד בין אופרטור השמה COPY CTOR ע"י פונקציות פרטיות
- **THE BIG THREE**: אם הגדרנו מחדש את אחד מהשלושה: אופרטור השמה, DTOR, COPY CTOR **בד"כ נידרש להגיד מחדש את שלושתם.** ונידרש לכך כמעט תמיד במחלקה שמכילה DATA MEMBERS שמוקצים באופן דינמי.

## דוגמאות CTORS:

| |
|---|
| Numbers c1,c2; /*ctor*/ |
| Numbers c1 = c2; /* copy ctor */ |
| c1 = c2; /* operator = */ |
| func(Numbers c1) /* copy ctor */ <-!!!COPY CTOR חשוב מאוד כשנקראת פונקציה נקרא ה<br>{Numbers func_var = c1}<-כי למעשה מתבצעת פה ההעתקה הסמויה |
| func(Numbers& c1) /* NOTHING – just referencing the name of the object */ |
| Numbers& c2 = c1 /* AGAIN NOTHING – just referencing the name of the object */ |
| Numbers c1 = Numbers(12); /*ctor – usually compiler optimization*/ |
| func(Numbers(12)); /*ctor – the same as the last one */ |
| Numbers func()<br>{<br>… return Numbers(12); /*ctor*/<br>} |
| Numbers func()<br>{<br>Numbers c1(15); /*ctor*/<br>return c1; /*copy ctor*/<br>} |
| Numbers& func()<br>{<br>Numbers c1(15); /*ctor*/<br>return *c1; /*error (either syntax or run time – c1 dies at the end of the function*/<br>} |
| Number (1.7 + c2) /*ctor + copy ctor*/<br>(ייצור טיפוס חדש) CTOR לביטוי בסוגריים<br>להשמה COPY CTOR |

## Inheritance

האפשרות לצור מחלקה נגזרת (יורשת) בעלת אותן תכונות כמו מחלקה אחרת (נורשת) בצרוף תכונות חדשות.

| הגדרת המחלקות: | טיפוסי המידע שנוצרים: |
|---|---|
| class the_base {<br>the_base(int i): … {…}<br>private: int a;<br>protected: int a1;<br>public: int b;<br>int method_a ();<br>}<br>class derive_from: the_base {<br>private:<br>char * d;<br>void method_b ()<br>} | the_base::a;<br>the_base::a1;<br>the_base::b;<br>the_base::method_a;<br><br>derive_from::a;<br>derive_from::a1;<br>derive_from::b;<br>derive_from::method_a();<br>derive_from::d<br>derive_from::method_b (); |

**כללים:**

- ברשימת האתחול של המחלקה היורשת נקרא הCTOR של המחלקה הנורשת כלומר חייבים להצהיר עליו (או להגדיר CTOR חסר ארגומנטים למחלקה הנורשת)
- (תזכורת:)private members עוברים בירושה אבל לא ניתן לגשת ל private members של המחלקה הנורשת מתוך החלקה היורשת.

- protected members עוברים בירושה וניתן לגשת ל protected members של המחלקה הנורשת מתוך החלקה היורשת.
- **CTOR,COPY CTOR,DTOR ואופרטור השמה אינם עוברים בירושה!**
- באתחול של מחלקה יורשת **נקרא הCTOR של מחלקת האם לפני יצירת שדה כלשהו של המחלקה היורשת. הריסת מחלקת האם נעשית אחרי הריסת המחלקה היורשת.**
- **אם אין למחלקה היורשת COPY CTOR ומעתיקים לתוכה אובייקט מסוג הBASE ייקרא הCOPY CTOR של הBASE**
- אם אנחנו קוראים לCOPY CTOR של המחלקה היורשת עם המחלקה היורשת **לא נקרא איזשהו CTOR** של ממחלקת האם
- אם אנחנו קוראים לCTOR של מחלקה יורשת עם עצם מסוג מחלקת האם ייקרא **הCOPY CTOR של מחלקת האם**
- אם ברצוננו לשכתב פונקציה במחלקה היורשת (OVERRIDE) בחלק מהמקרים נרצה להשתמש בפונקציה הנורשת. לדוגמא:

```
class derive_from: the_base {
public:
        void method_a ()
        {
                do-something
                the_base::a();
        }
}
```

## פולימורפיזם

אם אנו משתמשים במצביע לעצם מסויים ואנחנו רוצים לקרוא לMETHOD שלו אם המצביע הוא מסוג מצביע של מחלקת האם BY DEFAULT תיקרא הMETHOD של מחלקת האם.

```
derived_from obj;
the_base * obj_ptr = &obj;
obj->method_a();<- derived_from::method_a במקום ל the_base::method_a הקומפיילר יקרא ל
```

בכדי לעקוף זאת ניתן להשתמש במילה VIRTUAL:

```
class the_base {
public: virtual int method_a () = {…};
}
class derived_from {
public: virtual int method_a () = {…};
}
obj->method_a();<- derived_from::method_a כעת הקומפיילר יקרא ל
```

VIRTUAL נותן יכולת לקרוא בזמן ריצה לפונקציה שאותה אנחנו רוצים (טוב עבור מערך מצביעים לעצמי BASE שנפנה את חלקם להצביע על עצמי DERIVED ונעבור על המערך ונקרא לפונקציה האמיתית של כל עצם)

## Abstact classes-ו Pure Virtual

לעיתים אנו נרצה ליצור מחלקה **מופשטת** - זו מחלקה **שלא ניתן להתשמש בה אלא רק לרשת ממנה**
בכדי לעשות זאת נשתמש בפונקצית PURE VIRTUAL שהיא למעשה "חור" המכריח את המשתמש ליצור מחלקה יורשת ולממש את הפונקציה הזו.

```
class AbsClass {
. . .
        public:
. . .
        virtual void f() = 0; <- PURE VIRTUAL
}
class DeriveClass: public AbsClass {
. . .
        public:
. . .
        virtual void f() {do-something};
}

DeriveClass a; /* ok */
AbsClass b; /* compile error - Cannot create abstract object */
```

**דוגמא מוחשית:** אדם רוצה לקנות חיית מחמד אבל אין לו מה לעשות עם עצם מסוג Pet (לא קיים עצם כזה, כי זה מושג כולל אבל מופשט) אבל הוא יכול לקנות עצם מסוג Cat או Dog.

### Public Inheritance ("is-a" relationship) Vs. Private Inheritance ("implemented-as") relationship)

נגדיר "מחלקה אם" (base class):

```
class person {
private:
…
public: int age;
}
```

כאשר מתבצעת הורשת PUBLIC של מחלקת האם אזי המשתמש מחוץ למחלקה היורשת יכול לדעת שהמחלקה היא יורשת ממחלקת האם **ולהשתמש במחלקה היורשת עם המנשק של מחלקת האם** **מטרת הורשה PUBLIC היא למנוע שכפול קוד ולתת ממשק אחיד לטיפוסים שמשותפים למחלקה האם ולמחלקה היורשת**

```
class student: public person { /* the user knows that a student "is-a" person */
private:
…
public: int tuition_rate;
}

student a;
a.age = 24; /* that works ok */
```

כאשר מתבצעת הורשת PRIVATE של מחלקת האם אזי המשתמש מחוץ למחלקה היורשת לא יכול לדעת שהמחלקה היא יורשת ממחלקת האם **למעשה, כל הMEMBERS שהיו PUBLIC במחלקה האם הפכו להיות PRIVATE במחלקה היורשת –** לכן MEMBERS של המחלקה היורשת יכולים להשתמש בשדות אלה, **כי המחלקה יודעת יודעת שהיא יורשת מהמחלקה האם** (וממומשת עפ"י החוקים שלה - כלומר "implemented-as") **, אבל אף אחד מבחוץ לא יכול לדעת זאת.** **מטרת הורשה PRIVATE היא למעשה בעיקר למנוע שכפול קוד**

```
class professor: private person { /* the user doesn't know that a professor "is-a" person */
private:
…
public: int give_factor;
}

professor b;
b.age = 51; /* syntax error*/
```

# Template למנשק ADT

```
#ifndef MYADT_H
#define MYADT_H

/* if you are using an ADT that was taught you MUST NOT INCLUDE it here, but
include it
in the .c file - in order to avoid the revealing of your implementation*/

typedef int (*SomeElemFunc)(myADTElement);
/* in case there are functions that should be passed by the user (like copy,free
etc.)*/

typedef void* myADTelement

typedef struct myADT_t* myADT

typedef enum {TRUE=0,FALSE} bool;

typedef enum {SUCCESS=0,ADT_ELEMENT_NOT_FOUND, ADT_CANNOT_ALLOCATE,
ADT_ELEMENT_ALREADY_EXISTS, ADT_BAD_ARGUMENT, ...more results } Result;

/*A: In case the ADT holds the ACTUAL elements AND NOT COPIES of them */
Result MyADTCreate (myADT *);
/* returns SUCCESS and a pointer to the new ADT or ADT_BAD_ARGUMENT
if NULL was passed or ADT_CANNOT_ALLOCATE if case of memry allocation failure*/

/* OR: */
/*B: In case the ADT holds copies of the element
(note: the functions passed my differ according to the question's requirements)*/

Result MyADTCreate (myADT *,cmpFunc(MyADTElement,MyADTElement), MyADTElement
CpyFunc(MyADTElement), LblFunc(MyADTElement),FreFunc(MyADTElement)));
/* returns SUCCESS and a pointer to the new ADT or ADT_BAD_ARGUMENT
if NULL was passed or ADT_CANNOT_ALLOCATE if case of memry allocation failure*/

Result MyADTDestroy(myADT); /* returns SUCCESS or ADT_BAD_ARGUMENT if NULL was
passed */

Result MyADTAdd(myADT,MyADTElement(*)); /* returns SUCCESS or
ADT_ELEMENT_ALREADY_EXISTS ("SUCCESS" if using set.h or graph.h) or
ADT_CANNOT_ALLOCATE or ADT_BAD_ARGUMENT if NULL was passed */
Result MyADTRemove(myADT,MyADTElement(*)); /* returns SUCCESS or
ADT_ELEMENT_NOT_FOUND or ADT_BAD_ARGUMENT if NULL was passed */
Result MyADTIsElemIn(myADT,MyADTElement(*),bool *);/* returns SUCCESS and a value
associated with the boolean pointer or ADT_BAD_ARGUMENT if NULL was passed */
```

*** Should be used if the ADT holds the ACTUAL elements AND NOT COPIES of them**

# Class ל Template

**MYCLASS.H**

```
#ifndef MYCLASS_H
#define MYCLASS_H

template <class T, int S> // generic Class with max size
class MyClass {
public:
	MyClass(const T& val = T(), int size = 0); // C'tor - ALWAYS PREFERRED WITH
DEFAULT PARAM
	MyClass(const T* val = NULL, int size = 0); // C'tor for array - ALWAYS
PREFERRED WITH DEFAULT PARAM
	MyClass(const MyClass& p); // Copy C'tor
	T& operator[](const int & index); // set. For example - T[index] = 77;
	const T& operator[](const int& index) const; // get. int i = T[index];
	MyClass operator+ (const MyClass& right) const; // operator +: does not
change this & the right argument
	MyClass operator- (const MyClass& right) const; // operator -: same
	MyClass operator* (const MyClass& right) const; // operator *: same
	MyClass operator/ (const MyClass& right) const; // operator /: same
	MyClass& operator= (const MyClass& right);
	virtual ~MyClass(); // D'tor - ALWAYS PREFERRED VIRTUAL

private:
	T elements[size]; // an array of elements
	InitElements(T* elements, size);
}

#endif
```

**MYCLASS.CC**

```
template <class T, int S>
MyClass<T,S>::operator= (const MyClass<T,S>& right)
{
	if (this == &right) return *this; // check self assignement
	....
}
```

# דוגמאות למחלקות שלנמדו בתרגול

## STRING

### String.h

```
class String{
int len; // length
char* val; // value
void enter(const char*); // insert the string
public:
	String(); // c'tor for: String x;
	String(const char*);
// c'tor for: String x = "abc" or: String x("abc");
	String(const Sting&);
// copy c'tor for: String x = y or: String x(y)
	String& operator=(const String&);
// assignment operator for: x=y
	String& operator=(const char*);
// assignment operator for: x= "abc"
	virtual ~String(); // d'tor
	int len() const {return len};  // int i = x.len()
	const char& operator[](const int& i) const;
// char p = x[2]. const char& prevents a user change
	const char& operator[](const int& i);  // x[2] = 'a';
	String& operator +=(const String&); // x+=y
	int static AreEqual (const String& x, &const String& y)
// added be me to prevent friend comparison operators
	{return !(strcmp(x.val,y.val)))
	int operator==(const String&,const String&); // if (x==y)
	int operator!=(const String&,const String&); // if (x!=y)
	ostream& Print(ostream& os) const;
// added be me to prevent a friend << operator
	ostream& operator<<(ostream&,const String&);
}
```

### StringException.h

```
// Added by me.
#include <exception>
using std::exception;
class StingException: public exception {
public:
	virtual const char* what() const throw() = 0;
	/* Note: Pure virtual. We want only specific exceptions
	And we will use try catch (StringException)*/
};
class IndexOutOfRangeException: public StingException {
public: // This function describes the exception.
	virtual const char* what() const throw()
	/* throw() means, NO Exceptions are thrown by the "what" function*/
	{
		return "String: Index out of range";
	}
};
class StringOutOfMemoryException: public StingException {
public:
// This function describes the exception.
	virtual const char* what() const throw()
	/* throw() means, NO Exceptions are thrown by the "what" function*/
	{
		return "String: Out Of Memory";
	}
};
```

### String.cc

```
#include "String.h"
#include "StringException.h"

void String::enter(const char* st) throw (StringOutOFMemoryException)
```

```
{
      if (var) delete[] val; // if a string exists
      if (st==0) val = 0;           // Null pointer
      else
            try {
                        val= new char[strlen(st)+1])
                  }
            catch (bad_alloc&) {
                        val = 0;
                        len = 0;
                        throw StringOutOFMemoryException;
                  }
            strcpy(val,st);
            len = strlen(st);
}
String::String(): len = 0; val = 0; { }

String::String(const char*): len = 0; val = 0; enter(st); { }

String::~String { if (val) delete[] val; }

String& String::operator=(const char* st)
{
      enter(st)
      return *this;
}

String& String::operator=(const String& st)
{
      if (this!= & st) enter(st.val); // self assignment check
    return *this;
}
String::String(const String& st)
{
      len = 0;
      val = 0;
// set NULL so enter would not try to delete a null string and perform a null
reference
      enter(st);
}
char& String::operator[](int i) throw (IndexOutOfRangeException)
{
if ((i<0)|| (i> len))   throw IndexOutOfRangeException;
return val[i];
}

char String::operator[](int i) const throw (IndexOutOfRangeException)
{
if ((i<0)|| (i> len))   throw IndexOutOfRangeException;
return val[i];
}

char String::operator+=(const String & st) throw (StringOutOFMemoryException)
{
char *p;
if (st.len == 0) return *this;
      try {
            p = new char[len+st.len+1])
      }
      catch (bad_alloc&) {
                  throw StringOutOFMemoryException;
      }
      strcpy(p,val);
      strcat(p,st.val);
      enter(p);
      delete[]p;
      return *this;
```

```
}
int operator==(const String& x, const String& y) {return AreEqual(x,y);}
int operator!=(const String& x, const String& y) {return !AreEqual(x,y);}
ostream& operator<<(ostream& os, const String& st)
{
	return st.print(os);
}

ostream& String::print(ostream& ost) const
{
	return ost << val << endl;
}
```

# SET

## EXC.H (implements the SetExceptions class)

```
#ifndef _SET_EXC_H_
#define _SET_EXC_H_
// The SetException class inherits from std::exception.
#include <exception>
using std::exception;

class SetException: public exception {
public:
	// This function describes the exception.
	virtual const char* what() const throw()
	/* throw() means, NO Exceptions are thrown by the "what" function*/
	{
		return "The set is full!";
	}
};

#endif
```

## SET.H (implements the Set class)

```
#ifndef _SET_H_
#define _SET_H_

#include <iostream>
using std::ostream;
using std::istream;
#include "exc.h" // for exception handling

template <class T>
class Set {
private:
	int dim; // the size of the set
	int numElems; // current number of elements in the set
	T* buffer; // buffer holding set's data
	void copy(const Set& s);
	void clear();
	int find(const T& d) const;
public:
	Set(int s = 10);
	Set(const T* arr, int n);
	Set(const Set& s);
	Set(const T& d);
	virtual ~Set();
	Set& operator=(const Set& s);
	bool is_in(const T& d) const;
	bool insert(const T& d) throw (SetException); // throw only exceptions of
	type SetException
	bool erase(const T& d);
	bool full() const;
	bool empty() const;
	void clean();
	int size() const;
	Set& operator+=(const Set& v);
	Set& operator-=(const Set& v);
	bool compare(const Set& v) const;
	void print(ostream& ost) const;
	void scan(istream& ist);
};
```

**אופרטורים**

```
template <class T>
bool operator==(const Set<T>& v1, const Set<T>& v2);
template <class T>
bool operator!=(const Set<T>& v1, const Set<T>& v2);
template <class T>
ostream& operator<<(ostream& ost, const Set<T>& v);
template <class T>
istream& operator>>(istream& ist, Set<T>& v);
template <class T>
Set<T> operator+(const Set<T>& v1, const Set<T>& v2);
template <class T>
Set<T> operator-(const Set<T>& v1, const Set<T>& v2);
```

**פונקציות**

```
template <class T>
              void Set<T>::copy(const Set<T>& s)
{
      dim = s.dim;
      numElems = s.numElems;
      try {
            buffer = new T[dim];
      }
      catch (bad_alloc&) {    // catch by reference
      throw; // the exception is re-thrown (we have nothing to do)
      for (int i = 0; i < numElems; i++)
      buffer[i] = s.buffer[i];
}
template <class T>
void Set<T>::clear()
{
      if (buffer) { /* not null */
            delete[] buffer;
            buffer = NULL;
            dim = 0;
            numElems = 0;
      }
}
template <class T>
Set<T>::Set(int s)
: dim(s)
, numElems(0)
, buffer(NULL)
{
      try {
            buffer = new T[dim];
            }
      catch (bad_alloc&) {
      throw; // the exception is re-thrown
      }
}
template <class T>
Set<T>::Set(const Set<T>& s) /* copy c'tor */
{
      copy(s);
}
template <class T>
Set<T>::Set(const T& d) /*c'tor*/
{
      dim = 1;
      numElems = 1;
      try {
      buffer = new T(d);
      }
      catch (bad_alloc&) {
            throw; // the exception is re-thrown
      }
}
template <class T>
```

```
Set<T>::Set(const T* arr, int n) /*c'tor*/
{
	dim = n;
	numElems = 0;
	try {
	buffer = new T[n]; ¡par 147 }
	catch (bad_alloc&) {
		throw; // the exception is re-thrown
	}
		for (int i = 0; i < n; i++)
		insert(arr[i]);
	}
template <class T>
Set<T>::~Set()
{
 clear();
}
template <class T>
Set<T>& Set<T>::operator=(const Set<T>& s)
{
	if (this == &s)
	return *this;
	clear();
	copy(s);
	return *this;
}
template <class T>
Set<T>& Set<T>::operator+=(const Set<T>& s)
{
	if (this == &s) return *this; /* self assignment protection*/
	new_set(dim + s.dim);
	new_set.numElems = numElems;
	for (int i = 0; i < numElems; i++)
	new_set.buffer[i] = buffer[i];
	for (int i = 0; i < s.numElems; i++)
	new_set.insert(s.buffer[i]);
	*this = new_set;
	return *this;
}
template <class T>
Set<T> operator+(const Set<T>& s1, const Set<T>& s2)
{
	Set<T> s = s1;
	s += s2;
	196 return s;
}
template <class T>
Set<T>& Set<T>::operator-=(const Set<T>& s)
{
	if (this == &s)
	clean();
	for (int i = 0; i < s.numElems; i++)
	erase(s.buffer[i]);
	return *this;
}
template <class T>
Set<T> operator-(const Set<T>& s1, const Set<T>& s2)
{
	Set<T> s = s1;
	s -= s2;
	return s;
}
template <class T>
int Set<T>::size() const
{
	return numElems;
}
template <class T>
```

```
bool Set<T>::compare(const Set<T>& s) const
{
	if (this == &s) return true;
	if (numElems != s.numElems)
	return false;
	for (int i = 0; i < numElems; i++) {
	if (!is_in(s.buffer[i]))
	return false;
	}
	return true;
}
template <class T>
bool operator==(const Set<T>& s1, const Set<T>& s2)
{
	return s1.compare(s2);
}
template <class T>
bool operator!=(const Set<T>& s1, const Set<T>& s2)
{
	return !(s1 == s2);
}
template <class T>
void Set<T>::print(ostream& ost) const
{
	ost << "( ";
for (int i = 0; i < numElems; i++)
	ost << buffer[i] << " ";
	ost << ")";
}
template <class T>
ostream& operator<<(ostream& ost, const Set<T>& s)
{
	s.print(ost);
return ost;
}
	template <class T>
	void Set<T>::scan(istream& ist)
{
T d;
while (!ist.eof() && !full()) {
	ist >> d;
	insert(d);
	}
}
template <class T>
istream& operator>>(istream& ist, Set<T>& s)
{
	s.scan(ist);
return ist;
}


template <class T>
bool Set<T>::is_in(const T& d) const
{
	for (int i = 0; i < numElems; i++)
	if (buffer[i] == d) return true;
	return false;
}
template <class T>
bool Set<T>::full() const
{
	return (numElems == dim);
}
template <class T>
bool Set<T>::empty() const
{
	return (numElems == 0);
}
template <class T>
```

```
bool Set<T>::insert(const T& d)
{
	if (full()) throw SetException();
	if (!is_in(d)) {
		buffer[numElems++] = d;
		return true;
	}
	return false;
}
template <class T>
bool Set<T>::erase(const T& d)
{
	if (is_in(d)) {
		int i = find(d);
		for (int j = i; j < numElems; j++)
		buffer[j] = buffer[j + 1];
		numElems--;
		return true;
	}
	return false;
}
template <class T>
void Set<T>::clean()
{
	numElems = 0;
}
template <class T>
int Set<T>::find(const T& d) const
{
	for (int i = 0; i < numElems; i++)
	if (buffer[i] == d)
	return i;
	return -1;
}
#endif
```