ADT:

תכנות מודולארי:

- הלקוח רואה רק את הממשק (קובץ ה-header) ולא את פרטי המימוש. מקבל את קובץ ה-o. לאחר קומפילציה.
- ממשק: מזערי, מינימאלי ושלם.

ממשק: (קובץ ה-header)

- הצהרות על הפונקציות אותן מממש המודול.
- הגדרות של קבועים/טיפוסים אשר לקוח המודול צריך להכיר (למשל- ערכי שגיאה).

מימוש: (קובץ c.)

- מימוש הפונקציות שהוצהרו ב-header.
- פונקציות פנימיות למודול שלא נועדו לשימוש מחוץ למודול
- משתנים סטטיים של הקובץ.

הוראות include:

- עושים רק לדברים שצריך אותם במודול הספיציפי הזה.
- עושים ב-header רק אם יש צורך בהגדרות נוספות ב-header עצמו.

מטרה: לצמצם תלויות בין קבצים.

עקרונות-ADT:

- המבנה מוסתר משאר המודולים
  - אם משתנה מסוג ADT מקולקל- זה נגרם רק ע"י פונקציות המודול
  - שינוי מבנה ישפיע רק על הפונקציות של ה-ADT (ולא על מודולים אחרים).
- ל-ADT אסור להחזיר מצביעים לתוך הנתונים הפנימיים שלו- יגרום לחשיפת המימוש ויאפשר ללקוח לשנות את ה-ADT ולהעבירו למצב שאינו תקין.
- עדיף לא להשתמש בפונקציות get ו-set רבות מידי, אלא להעביר טיפול לרמה נמוכה יותר.
- טיפוסים צריכים להיות ממומשים כיחידים.
- פונקציות ADT צריכות להחזיר חווי על הצלחה או שגיאות שונות.
- הימנעות משכפול קוד בפונקציות create ע"י קריאה ל-destroy בכישלון הקצאה בסוף.

©ענת עציון & עדי פוקס

Template למנשק ADT של מערכת

```
#ifndef MYADT_H
#define MYADT_H

#include "needed_types.h"

typedef enum {TRUE,FALSE} Bool;

typedef double Scalar; // optional- for better readability

typedef const char *Pid // optional- for better readability

typedef struct myADT_t *MyADT;

typedef enum {SUCCESS, FAIL, OUT_OF_MEMORY, NULL_ARGUMENT,
ELEMENT_NOT_FOUND, ELEMENT_ALREADY_EXISTS,… more results }
MyADTResult;

/* in case another struct is described and it's type is needed for
the ADT */
typedef struct type_t {
      int field1;
      char* field2;
}Type;

MyADT myADTCreate ();
/* returns a pointer to the new ADT or NULL when fails to allocate
memory */

void myADTDestroy(MyADT system);
/* like free is a void function so is the destroy, do nothing
(return;) if a NULL pointer is sent*/

MyADTResult myADTAdd(MyADT system, const char* name,… more details);
/* returns: NULL_ARGUMENT, OUT_OF_MEMORY- failed to allocate,
ELEMENT_ALREADY_EXISTS or SUCCESS */

MyADTResult myADTRemove(myADT system, int id, …other details);
/* returns: NULL_ARGUMENT, ELEMENT_NOT_FOUND or SUCCESS*/

MyADTResult myADTGetDetail (myADT system, int* id, char** name, Bool*
match);
/* returns: NULL_ARGUMENT, ELEMENT_NOT_FOUND or SUCCESS
and changes the pointers to a copy of the data (must not return the
real data- encapsulation!)*/

MyADTResult myADTGetCollection(myADT system, int id, SpecialType**
elements, int* elementSize);
/* returns: NULL_ARGUMENT, ELEMENT_NOT_FOUND or SUCCESS and changes
the pointer (elements) to point to the collection and returns the
collection size */

#endif
```


©ענת עציון & עדי פוקס

לשים-♥:

- האם משתנה נדרש להיות int או double.
- לא לשכוח לכתוב התרעות אפשריות לאחר כתיבת הצהרות הפונקציות.
- אם ישנו טיפוס נוסף, לא בהכרח צריך לעשות include, אם הוא מפורט ניתן להגדירו כאן.
- אם נאמר כי אובייקט מזוהה רק לפי שמו למשל, אזי השם (const char) הינו הדבר היחיד שמועבר לפונקציות ושאר המימוש יהיה בקובץ c.
- אם נאמר כי זהות מורכבת משני סטרינגים, ניתן לקרוא להם במשתנה אחד- יותר נוח.

Template ל-מבנה נתונים של מערכת:

```
#include "myADT.h"
#include "set.h" / "linked_list.h"

/* The struct of the ADT */
struct MyADT_t {
	Set people;
	LinkedList workers;
};

/* Internal struct, (not ADT) */
typedef struct Worker_t {
	char* name;
	int id;
} Worker;

/* Internal struct, (not ADT) */
typedef struct WorkerImage_t {
	int id;
	Image image; // (include to "image.h" was done in MyADT.h)
} WorkerImage;
```


©ענת עציון & עדי פוקס

Template למימוש פונקציה:

```
MyADTResult functionName(MyADT system, double param, Bool* match,
char** name, Element** newCollenction) {

   /* initializing params */
   double param;
   Element *currentElem;
   Element2 *currentElem2;
   char* tmpName = NULL;  // when needs to return a copy of a string
   Bool *match = FALSE;
   Element3 newElem;
   myADTResult result;

   /* check for NULL arguments */
   if (system == NULL || match == NULL || name == NULL )
      return NULL_ARGUMENT;

   /* check for other specific illegal arguments */
   if (param < 0.0)
      return ILLEGAL_PARAM;

   /* iterating over two sets */
   SET_FOREACH(currentElement, system->elementsSet) {
      SET_FOREACH(currentElement2, currentElement->elements2Set){
            if (currentElem2->param == param){
                  tmp_name = currentElement->name;
                  *match = TRUE;
            }
      }
   }

   /* creating a new copy of the string for retrieval */
   if (tmpName != NULL) {
      *name = (char*)malloc(strlen(tmpName)+1);
      if (*name == NULL)
            return OUT_OF_MEMORY;
      strcpy(*name, tmpName);
   }

   /* creating a new collection for retrieval */
   *newCollection = (Element*)malloc(sizeof(struct Element_t)*num);
   if (*newCollection == NULL)
      return OUT_OF_MEMORY;
   for (i = 0; i < num; i++) {
      …
      (*newCollection)[i] = newElement;
   }

   /* using a Set function (can be used different than learned */
   if (SetAdd(system->elemSet, param ,&newElem) != SUCCESS)
      return FAILURE;

   return SUCCESS;
}
```


©ענת עציון & עדי פוקס

לשים-♥:

- לא להחזיר מצביע לנתונים פנימיים. להחזיר עותק שלהם. וביצירה של אובייקט חדש, להעתיק את הנתונים ולצרף רק את העותק שלהם לאובייקט החדש.
- לשים-לב למימוש המדויק של מבנה הנתונים ולעבוד עם הגישות <- בהתאם.
- לא לשכוח להחזיר SUCCESS בסוף.
- להרוס עם פונקצית destroy או free אובייקטים זמניים שנוצרו, לשים-לב בעיקר בלולאה להרוס בכל איטרציה איבר שאולי נוצר באיטרציה הקודמת (ניתן להשתמש ב-flags כדי לדעת האם נוצר איבר זמני באיטרציה הקודמת שיש צורך להרוס).
- ניתן לכתוב assert (result == SUCCESS) לאחר פונקציות החזירות ערך Result.
- ניתן להשתמש ב-Set לדוגמא עם פונקציות מעט שונות מהפונקציות שנלמדו בכיתה (כל עוד מסבירים את השינויים).

©ענת עציון & עדי פוקס

Template למנשק ADT של מבנה נתונים:

(המבנה שומר את האובייקט ולא העתק שלו)

```
#ifndef MYADT_H
#define MYADT_H


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;

Result MyADTCreate (myADT *);

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 */

#endif
```


©ענת עציון & עדי פוקס

דוגמא ממבחן- עץ בינארי:

```
typedef struct BT_t* BT;
typedef void* Element;
typedef enum{FAIL SUCCESS} Result;
typedef enum{TRUE, FALSE} Bool;

Result BTCreate(BT* root);
Result BTDestroy(BT root);
Result BTAddNode(BT root, long int key, Element d);
Result BTDeleteNode(BT root, long int key);
Bool BTNodeExists(BT root, long int key);
Element BTGetNodeData(BT root, long int key);


struct BT_t {
   Element el;
   long int key;
   struct BT_t* left;
   struct BT_t* right;
};
```

שימוש במבנה הנתונים שנכתב:

```
struct Lib_t {
   BT bt;
}
typedef struct Lib_t* Lib;
typedef enum{SUCCESS, FAIL} LibResult;

struct Book_t {
   char* name;
   long int key;
   int is_bor;
} Book;

Book CreateBook(long int book_id, char *book_name) {
   Book book=(Book)malloc(sizeof(struct Book_t));
   if (book == NULL)
         return NULL;
   book->name = (char*)malloc(strlen(book_name)+1);
if (book->name == NULL) {
   free(book);
   return NULL;
}
strcpy(book->name, book_name);
book->key = book_id;
book->is_bor=0;
return book;
}

void DestroyBook(Book book) {
   if (book == NULL)
         return;
   free(book->name);
   free(book);
}
```


©ענת עציון & עדי פוקס

SET:

מיועד למימוש אוסף של איברים שאין בהכרח חשיבות לסדר ביניהם או קשר כלשהו אחר ביניהם. כל איבר מופיע ב-Set פעם אחת (לא יוסיף איבר שכבר נמצא בקבוצה, ולא יחזיר הודעת שגיאה).

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 הפנימי של Set.

SetIsEmpty(s) האם הקבוצה S ריקה

פונקציות:

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

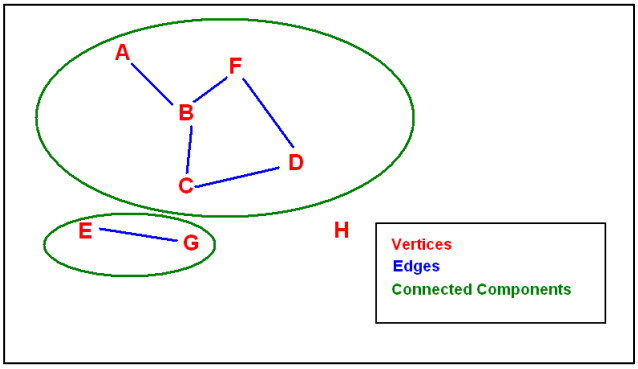
*** אנו יוצאים מנקודת הנחה שלשתי הקבוצות אותם אלמנטים ואותן פונקציות טיפול עבור כל אלמנט**

©ענת עציון & עדי פוקס

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)

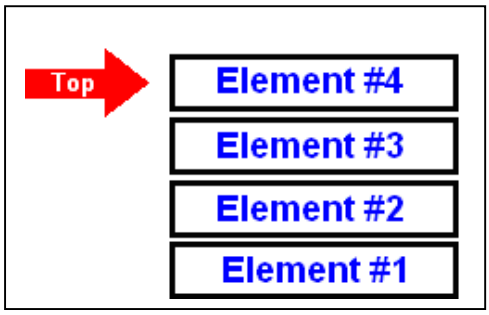
** שימוש רקורסיבי בתוצאה ובקבוצה שמועברת כפרמטר לפונקציה DEPTH FIRST TRAVERSAL שממומשת באופן רקורסיבי מציאת שכן לכל שכן (ובדיקה שלא הוספנו אותו כבר לקבוצה) .

©ענת עציון & עדי פוקס

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:

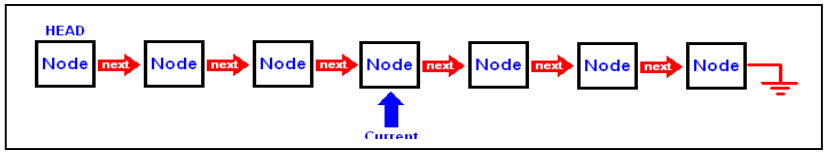
מה הפונ' עושה	מה מוחזר	שם
יוצרת את המחסנית.	**מחסנית חדשה** בגודל מקסימלי של MaxSize	Stack **create**(int max_size, cpy_func, free_func)
הורסת את המחסנית	-	Void **destroy**(Stack)
דחיפת איבר לראש המחסנית	**FAIL** אם האיבר או מצביע המחסנית הם NULL. **SUCCESS** במקרה של הצלחה	Result **push** (Stack, Element)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

©ענת עציון & עדי פוקס

LinkedList:

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 לLinkedList_t STRUCT

KeyForListElement טיפוס איבר הסינון (עבור פונקצית הסינון) מסוג *void - אפשרי להשתמש בו בתור הערך שלפיו תסונן הרשימה, ניתן להעביר אותו לרשימה בצירוף מצביע לפונקצית הסינון כדי שתשתמש בו. הפונקציה החדשה תחזיר TRUE רק עבור כל איבר שהתאים למפתח.

copyListElemFunc מצביע לפוקנצית ההעתקה שמקבלת ומחזירה ListElement (ניתן להחזיר עותק של האיבר המועבר או להחזיר מצביע אליו)

freeListElemFunc מצביע לפוקנצית השחרור שמקבלת ListElement ומשחררת אותו (ניתן להעביר פונקציה שלא עושה כלום ואז לא ישוחררו האיברים)

printListElmFunc: מצביע לפונקצית ההדפסה שמקבלת stream להדפסה ו-ListElement ומדפיסה אליו סטרינג של האובייקט.

matchListElemFunc מצביע לפוקנצית הסינון שמקבלת ListElement וKeyForListElement

ומחזירה 1 אם האיבר תואם את המפתח או 0 אם האיבר לא תואם את המפתח (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

ListResult טיפוס enum של

{LIST_SUCCESS, LIST_FAIL ,LIST_BAD_ARGUMENTS, LIST_OUT_OF_MEMORY}

פונקציות:

מה הפונ' עושה	מה מוחזר	שם
יוצרת רשימה חדשה ומשייכת לכל האיברים פונקציות העתקה ושחרור	**מצביע** לרשימה החדשה (או NULL במקרה של כשלון).	LinkedList **linkedListCreate**(copyListElemFunc, freeElemFunc, printListElemFunc)
הורסת את הרשימה	**LIST_SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL	ListResult **linkedListDestroy**(LinkedList)
מחזירה את מספר האיברים ברשימה	**מספר האיברים** ברשימה	int **linkedListGetNumElements**(LinkedList)
הכנסת איבר לראש הרשימה	**LIST_SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL	ListResult **linkedListInsertFirst**(LinkedList ,ListElement)

©ענת עציון & עדי פוקס

	LIST_OUT_OF_MEMORY כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL	
ListResult **linkedListInsertBeforeCurrent**(LinkedList , ListElement)	**LIST_SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_OUT_OF_MEMORY** כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL	הכנסת איבר לפני האיבר הנוכחי
ListResult **linkedListInsertAfterCurrent**(LinkedList, ListElement)	**LIST_SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_OUT_OF_MEMORY** כשלון בהקצאה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL	הכנסת איבר אחרי האיבר הנוכחי
ListResult **linkedListGetCurrent**(LinkedList, ListElement*)	**LIST_SUCCESS** במקרה של הצלחה. **בנוסף בפרמטר יוחזר המצביע לאיבר הנוכחי.** **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_FAIL** הרשימה שהועברה היא ריקה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL.	החזר את האיבר הנוכחי המוצבע במסחנית
ListResult **linkedListRemoveCurrent**(LinkedList)	**LIST_SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_FAIL** הרשימה שהועברה היא ריקה או שפונקצית ההעתקה שהועברה בCREATE החזירה NULL.	הסר את האיבר הנוכחי המוצבע במסחנית
ListResult **linkedListGoToHead**(LinkedList)	**LIST_SUCCESS** במקרה של הצלחה. **תוצר לוואי: האיטרטור יצביע לאיבר הראשון ברשימה** **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL	לך לאיבר בראש הרשימה
ListResult **linkedListGoToNext**(LinkedList)	**LIST_SUCCESS** במקרה של הצלחה. **תוצר לוואי: האיטרטור יצביע לאיבר הבא ברשימה** **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_FAIL** במקרה שהרשימה ריקה או שהאיבר הנוכחי היה האחרון	לך לאיבר הבא ברשימה
ListResult **linkedListFind**(LinkedList, KeyForListElement, matchListElemFunc)	**LIST_SUCCESS** במקרה של הצלחה. **תוצר לוואי: האיטרטור יצביע למופע הראשון ברשימה של האיבר שהועבר** **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_FAIL** במקרה שהרשימה ריקה או שהאיבר לא נמצא ברשימה	מציאת איבר והעברת האיטרטור הפנימי כך שיצביע עליו.
ListResult **linkedListFilterElements**(LinkedList, LinkedList*, matchListElemFunc, KeyForListElement);	**SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL **LIST_OUT_OF_MEMORY** כשלון בהקצאה.	מקבל רשימה, פונקציית match והערך לפיו ה-match יעבוד (key) ומחזיר רשימה חדשה אחרי פילטר
ListResult **linkedListPrint**(LinkedList, FILE*, int)	**SUCCESS** במקרה של הצלחה. **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL	מקבל רשימה, מספר איברים להדפסה וערוץ להדפיס אליו ומדפיס את הרשימה.
ListResult **LinkedListSortElements**(LinkedList cmpListElemFunc)	**LIST_SUCCESS** במקרה של הצלחה. (והרשימה שמועברת כפרמטר ממוינת) **LIST_BAD_ARGUMENTS** במקרה שמועבר מצביע NULL.	ממיינת את הרשימה

©ענת עציון & עדי פוקס

C-SHELL:

תחילת הסקריפט: `#! /bin/tcsh –f`

עבודה עם קבצים:

pwd	קבלת מיקום נוכחי של ה-path של המדריך הנוכחי
ls	הדפסת הקבצים במדריך הנוכחי
ls subdir	הדפסת הקבצים בתת-תיקייה subdir
ls -1	כמו ls רק שכל שם קובץ, בשורה נפרדת
ls -p	משרשר תווים, לפי הסוג: / תיקייה, כלום לקובץ
ls –l	הדפסת הקבצים במדריך הנוכחי עם יתר פירוט

```
-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
```

foreach F(*) ** echo $F** **end**	מעבר על כל הקבצים במדריך

if (-d (-f) $F) then
…
endif
	בדיקה האם זהו מדריך (או קובץ)

rm <filename>	מחיקת קובץ
command !>> file	כתיבה לקובץ (אם הוא לא קיים, יוצרים אותו)

סקירת תוכן קבצים

cat (-n) file1 file2 file3	הדפסת תוכן קבצים (ומספור שורות) לפלט הסטנדרטי
head -n file1 file2 file3	הדפסת n שורות ראשונות של קבצים לפלט הסטנדרטי
tail -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-n

Pipelining:

head -3 src_file | tee out1 out2| tail -2	שכפול הפלט לקבצים נוספים

מיון שורות קבצים-sort:

Sort [options] (files)

-b	מתעלם מרווחים בתחילת שורה
-d	מסדר בסדר מילוני (מתעלם מסימני פיסוק)
-f	מתעלם מ-lower/uppercase
-m	ממזג קבצים ממוינים לקובץ אחד
-n	ממיין לפי ערך מספרי מהקטן לגדול
-o file	הדפס פלט לקובץ file
-r	הופך את סדר המיון
-u	מסנן שורות זהות
+n	מדלג על n המילים הראשונות לפני ביצוע המיון
-k num	מתייחס לכל שורה החל ממילה num (ממוספר מ-1)

©ענת עציון & עדי פוקס

הדפסת שורות לפי הופעת סטרינג- grep:

grep [options] [^]word [files]

מדפיס לפני כל שורת פלט את מספק הבלוק שלה	**-b**
מדפיס רק את מספר השורות שנמצאו	**-c**
מדפיס את השורות עצמן ללא שמות הקבצים בהם הן נמצאות	**-h**
מתעלם מ-lower/uppercase	**-i**
מדפיס את שמות הקבצים ללא השורות	**-l**
מדפיס את השורות ואת מספרן בקבצים	**-n**
מדפיס את השורות בהן לא מופיעה המחרוזת	**-v**
מדפיס את השורות בהן מופיעה word בדיוק ולא כתת מחרוזת	**-w**
מדפיס את השורות שמתחילות ב-word	**^word**
מדפיס שורות שבהן מופיע ביטוי של יותר ממילה אחת	**"word1 word2"**

סינון שורות זהות-uniq:

uniq [options] [input [output]]

מורידה מהקובץ output שורות זהות סמוכות (ושולחת ל-output).

מדפיס כל שורה פעם אחת וסופר עותקים לכל שורה	**-c**
מדפיס רק שורות המופיעות יותר מפעם אחת	**-d**
מדפיס רק שורות המופיעות פעם אחת בדיוק	**-u**
מתעלם מ-n המילים הראשונות (לפני החלטה אם שורות זהות)	**-n**
מתעלם מ-lower/upper case	**-i**

הדפסת תווים/שדות מכל שורה-cut:

cut options [files]

מדפיס תווים או שדות מכל אחת משורות הקלט

מוציא קבוצת תווים לפי האינדקסים המופיעים ב-list	**-c list**
מוציא קבוצת שדות לפי האינדקסים המופיעים ב-list שדות: קבוצות תווים המופרדות ע"י תו מפריד (default: <tab>)	**-f list**
בנוסף ל f– מאפשר לקבוע את "c" כתו המפריד	**-d "c"**

התו המפריד קיים בשורה, אך ניתן מספר שדה שאינו קיים: תודפס שורה ריקה.

התו המפריד לא קיים בשורה: תודפס כל השורה

```
> cat file1
a11 a12 a13 a14 a15
b21 b22 b23 b24 b25
c31 c32 c33 c34 c35
> cut -c1-3,5,8-10 file1
a11a a1
b21b b2
c31c c3
> cut -d" " - f2,4 file1
a12 a14
b22 b24
c32 c34
```

```
> cat file2
a11:a12:a13
b21:b22
c31 c32 c33
> cat file2 | cut -d":" -f1
a11
b21
c31 c32 c33
> cat file2 | cut -d":" -f3
a13

c31 c32 c33
```

ספירת מילים-wc:

wc [options] [files]

מדפיס את מספר התווים, המילים או השורות בקלט

מדפיס את מספר התווים בלבד	**-c**
מדפיס את מספר השורות בלבד	**-l**
מדפיס את מספר המילים בלבד	**-w**

- אם לא ניתן flag, מודפס בפורמט: <# lines> <# words> <# chars> filename
- אם יש כמה קבצי קלט, מודפס גם סיכום של כולם.

©ענת עציון & עדי פוקס

הדפסה לפי פורמט-printf: printf format strings

הדפסת המחרוזת string לפי הפורמט format אשר יכול להכיל תווים רגילים או תווי עריכה.

%s	מדפיס את המחרוזת
%X.Y	מדפיס מחרוזת שכולה באורך X כאשר יש הגבלה ל-Y תווים מתוך כל מחרוזת ספיציפית ב-strings.

תבנית לשמות קבצים (ביטויים רגולריים):

*.c	כל הקבצים עם סיומת c
AB	קבצים המכילים את צירוף האותיות AB
prog?	מתחיל ב-prog ואחריו יש בדיוק תו אחד
file[13]	מתחיל ב-file ואחרי אותיות אלו מופיע או 1 או 3
file[A-Za-z13]	מתחיל ב-file ואחרי אותיות אלו מופיעה אות מהא"ב האנגלי או אחת מהספרות 1 או 3
x{aa,bb,cc}y	מכיל בין x ו-y אחת מן המחרוזות

עבודה עם משתנים:

set <variable> = <value>	יוצר משתנה variable עם ערך value
echo $var	הדפסת תוכן משתנה var ($ יתן את ערך המשתנה)

משתנים מספריים:

@ j = 2+ $i	ביצוע פעולות חשבון, שמים @ ורווח בתחילת השורה
$2-\underbrace{3+4}_{7}=-5$	אסוציאטיביות פעולות לא סטנדרטית (לכן אפשר להשתמש בסוגריים).
@ i = 3	השמת מספר נחשבת גם היא כפעולת חשבון
@ i++	הגדלת i ב-1 (כמו ב-C)
@ i = $i / 3	תוצאת החילוק תהיה שלמה ותעוגל למטה

- הפעלת פעולת חשבון על מחרוזת תגרור שגיאה
- פעולות חשבון נעשות רק על מספרים שלמים.

מחרוזות ורשימות:

set A = "abc def"	מחרוזת המכילה יותר ממילה אחת, תוחמים ב"
set list = (123 yom tov)	יצירת רשימה, תוחמים ב-()
set list[n] = layla	ערך האיבר ה-n ברשימה (מספור איברים מתחיל מ-1)
$#list	החזרת מספר האיברים ברשימה
$list[-n]	תת-רשימה מהאיבר הראשון עד לאיבר ה-n
$list[m-]	תת-רשימה מהאיבר ה-m עד לסוף הרשימה
$list[m-n]	תת-רשימה מהאיבר ה-m עד לאיבר ה-n
$list[*]	קבוצה המורכבת מכל איברי הרשימה
shift list echo $list -> yom tov	הזזת איברי הרשימה מקום אחד שמאלה ודריסת האיבר הראשון
${list}/$f	רשימה פחות איבר

©ענת עציון & עדי פוקס

נקודות מתקדמות:

התייחסות למשתנה כרשימה בת איבר אחד	**set a = "abc def"** **echo $a[1] -> abc def**
פירוק מחרוזת לרשימת מילים – השמה בתוך סוגריים	**set b = ($a)** **echo $b[2] -> def**
סוגריים מסולסלים להדגשת שמו של משתנה למשל כאשר רוצים להצמיד מחרוזת לערך של משתנה	**set b = 123** **set a = ${b}4** **echo $a -> 1234**

עיבוד פקודה-סוגי גרשיים:

backticks: ניתן לשתול פקודה בתוך מחרוזת/פקודה אחרת: מקיפים את הפקודה נשתלת בגרשיים הפוכים. **הפקודה תתבצע לפני שאר השורה ותוחלף בערך המוחזר ממנה**:

```
> echo you have `ls | wc -l` files in `pwd`
you have 43 files in /home/rotics
```

גרשיים כפולים: מאפשרים **החלפת משתנה בערכו** (בגרש יחיד המשתנה לא תתבצע ההחלפה).

```
> set a = AAA
> echo "a=$a"
aAAA
> echo 'a=$a'
a=$a
```

- החלפת שמות קבצים (עם ביטויים רגולריים) לא מתבצעת בתוך גרשיים כפולים או יחידים.

backticks כרשימה:

1. אם ה-backticks **לא בתוך גרשיים כפולים**, כל רווח תו או סימן מעבר שורה, מפריד בין האיברים ברשימה.
2. אם ה-backticks **בתוך גרשיים כפולים**, רק סימן מעבר שורה מפריד בין האיברים ברשימה.

```
> cat lines
a b
c d
> set A = `cat lines`
> echo $#A
4
> echo $A[1]
a
> echo $A[2]
b
```

```
> set A = "`cat lines`"
> echo $#A
2
> echo $A[1]
a b
> echo $A[2]
c d
```

לולאות:

repeat n <command>	מבצע את הפקודה n פעמים
while <condition> **<commands>** **end**	מבצע בלוק כל עוד תנאי מתקיים
foreach var (1 2 3) **echo $var** **end**	מתבצע עבור כל איבר ברשימה

©ענת עציון & עדי פוקס

תנאים:

if עם פקודה אחת בלי else	**if (<expression>) <one command>**

if עם כמה פקודות ואופציה ל-else	**if (<expression>) then**
	<commands>
	[else
	<commands>
	endif

סוגי תנאים בפקודת –if:

השוואת מחרוזות רגילה	**if ($var == "abc")**
התאמה חלקית בין מחרוזות בצד ימין יכולה להופיע תבנית מחרוזת הכוללת ביטוי רגולרי. שלילה של ~= היא: !~	**if (str1 =~ str2)**
השוואת מספרים	**if ($var > 10)**
משוערך ל-true אם תיקייה-dir1 קיימת במיקום הנוכחי	**if (-d dir1)**
משוערך ל-true אם קובץ-file1 קיים במיקום הנוכחי	**if (-f file1)**

אופרטורים להשוואה:

<, >, =<, => משווים בין **מספרים**

==, =!, =~, !~ משווים בין **מחרוזות**

אופרטורים לוגיים:

!, &&, || (כמו ב-c)

שימוש בסוגריים (כמו ב-c)

Operator	What it means
()	change precedence
~	complement
!	negation
*/ %	multiply, divide, modulo
+ -	add, subtract
<< > >	left shift, right shift
<= >= < >	relational operators
== != =~ !~	string comparison/pattern matching
&	bitwise AND
^	bitwise exclusive OR (XOR)
|	bitwise inclusive OR
&&	logical AND
||	logical OR

פקודות נוספות:

אוסף של תנאים (switch)	**switch (<variable or string>)** **case <pattern1>:** **<commands>** **breaksw** **case <pattern2>:** **<commands>** **breaksw** **[default:** **<commands>]** **endsw**
קפיצה למקום המסומן ב-label (לא מומלץ)	**goto label**

©ענת עציון & עדי פוקס

העברת פרמטרים לסקריפט:

הפקודה עצמה	**$0**
ערך הארגומנט הראשון שמועבר לסקריפט	**${1} / ${argv[1]}**
רשימת כל הארגומנטים	**$***
מספר הארגומנטים שהועברו לסקריפט (לא סופר את הפקודה עצמה).	**${#argv}**

קריאת שורות מקבצים:

קריאת שורה מה-standart input **כרשימה** למשתנה-line (כל מילה הינה איבר ברשימה).

- ללא הסוגריים "$<" = set line ישים במשתנה line שורה מהקלט **כמחרוזת** (פחות שימושי).
- אין אפשרות לקרוא ישירות מקבצים אלא רק מה-standart input (בתוך סקריפט).

```
set line = ($<)

while (${#line}!=0)

... set line = ($<)   ← קידום לשורה הבאה

end
```


©ענת עציון & עדי פוקס

דוגמאות סקריפטים ב-C-shell:

```
script name:
#! /bin/tcsh -f

# שגיאה- מספר פרמטרים לא נכון לסקריפט
if($#argv != 3) then
     echo Wrong number of parameters
     exit 1
endif

# שגיאה- שימוש ב-flag לא נכון
if("$2" != "-name") then
     echo Second parameter must be -name
     exit 2
endif

# קבלת פרמטרים שניתנו כקלט לסקריפט
set dir = "$1"
set pattern = "$3"

# שגיאה- התיקייה שניתנה לא קיימת
if(! -d "$dir") then
     echo First must be an existing directory
     exit 3
endif

# ניתן להפעיל סקריפט באופן ישיר ואין צורך לשים את הפלט במשתנה
rec_find "$dir" "$pattern" | sort | summary_find

# סקריפט יכול לקבל קלט משני מקומות: ארגומנט ישיר (argv) או-pipe
set common_words =(`statistics $file | hpf $max_occur`)

# מעבר על כל הקבצים וכתיבת 4 האותיות הראשונות של כל אחד לתוך temp_file
foreach f(*)
     if(-f $f) then
          set a= `echo $f | cut -c1-4`
          echo $a >>! temp_file
     endif
end

# הדפסה כפלט של איברים מתוך line, יודפסו בשורה אחת עם רווחים ביניהם
set line = ($<)
while(${#line} != 0)
     set grade = `cat *.grades | grep $line[1] | CalcOne`
     echo $line[1] $line[2] $line[3] $grade
     set line = ($<)
end
```


©ענת עציון & עדי פוקס

```
# מעבר על כל איברי הרשימה
foreach arg($list)


# שימוש בקלט של הסקריפט כרשימה שממנה ניתן להדפיס פלט הלאה
set num_links = `cat $page | Get | sort -f -u | Count`

# שימוש בפונקציה printf בדומה ל-C ושימוש ב / כדי לסמן ששורה ממשיכה
if ($num_links[1] != 0) then
     printf "%s: %s broken links out of %s\n" \
           $page $num_links[1] $num_links[2]
endif

# כדי להדפיס ערכי משתנים וטקסט אחריהם, יש להשתמש ב- ${משתנה}טקסט$
echo "$param1 \t ${param2}:$param3 more words"

# מספר הפרמטרים אינו ידוע ורוצים לקבל את כל הפרמטרים ממספר-2 עד הסוף לתוך רשימה. אז
קוראים את $* ואז מסירים עם shift את הראשון
set file_name = ${1}.align
set len = ($*)
shift len

# רעיון יפה- אתחול של רשימה של אפסים כדי מאוחר יותר למצוא את המקסימום של כל אחד
set length = ()
set i = 1
while ($i <= $num_of_fields)
     set length = (0 $length) # update the list- adding 0
     @ i++
end


# קביעת גודל הסטרינג שיודפס ב-printf (יהיה ריפוד ברווחים אם צריך)
printf "%${actual_length}s" $line[$i] >>! $file_name
```


©ענת עציון & עדי פוקס

C++:

Const:

- טיפוס של קבוע. משתנים מסוג const חייבים **לאתחל**. לא ניתן לבצע **השמה** אליהם.
- ניתן להגדיר מצביעים לטיפוסים קבועים: const int* pi, משתנה שיכול להחזיק כתובות של משתנים מסוג: const int.
- ניתן להמיר int* ל-const int* אך לא להפך.
- ניתן להגדיר כ-const פרמטרים וערכי החזרה מפונקציות (חוסך בתיעוד).

Reference:

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

```
int i = 8;
int &j = i;
```

- מרגע ההגדרה כל השמה למשתנה אחד תשנה את ערכי שני המשתנים.
- חובה להגדיר משתנה מתייחס באתחולו (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!

העברת פרמטר By Reference לפונקציה:

- כאשר רוצים שהפונקציה תשנה משתנים מחוצה לה.
- אובייקטים גדולים- לא רוצים להעביר by value ולהעתיק.

```
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 x = index(a,7); ← x = a[7]
```

Const Reference (עצם זמני שהפונקציה מתחייבת לא לשנות) :

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

טעות נפוצה: החזרת reference למשתנה לוקאלי - יפסיק להתקיים עם היציאה מהפונקציה.

©ענת עציון & עדי פוקס

הבדלים בין reference למצביע:

אתחול	reference חייב להיות מאותחל	מצביע לא חייב להיות מאותחל
ערך NULL	אין ל-reference ערך NULL, הוא חייב להתייחס למשתנה כלשהו	מצביע יכול להכיל ערך NULL
שינוי הצבעה/התייחסות	reference יכול להתייחס למשתנה אחד בלבד	מצביע יכול לשנות משתנה שהוא מצביע אליו

זיכרון דינאמי:

הקצאת אובייקטים בודדים	**new type**
שחרור אובייקטים בודדים	**delete pointer**
הקצאת מערך בגודל n	**new type[n]**
שחרור מערך	**delete[]**
הקצאת מערך דו-מימדי	**int** array** **array = new int*[5]** **for (int i = 0; i < 5; i++)** ** array[i] = new int;**
שחרור מערך דו-מימדי	**for (int i =0; i < 5; i++)** ** delete array[i];** **delete[] array;**

- new לא מחזירה NULL עם כישלון (לכן לא צריך לבדוק אם חזר NULL).
- delete לא יעשה כלום אם יפעל על NULL (לכן לא צריך לבדוק שזה לא NULL).

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)

סדר ההמרה:

1. המרה מדויקת + 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.

- הפונקציות לא נבדלות ע"י סוג הערך המוחזר.
- אסור שיהיו שתי פונקציות עם אותו שם ואותם פרמטרים.

Namespaces:

- הרחבה לבלוקים של קוד, ניתן יהיה להגדיר ולהשתמש ב namespace אחד בכמה קבצים
- עדיף להצהיר על פונקציה ספציפית- **Using namespace1::func1** כדי להימנע מהתנגשות בין שמות ולהימנע מניפוח של הקובץ בגלל טעינת רכיבים מיותרים.

©ענת עציון & עדי פוקס

Default parameters:

אם לא מסופק פרמטר לפונקציה, אז הפונקציה תקבל את הערך הדיפולטי.

ניתן לתת ערכי ברירת-מחדל רק לפרמטרים האחרונים

```
int f1(int a = 0, int b = 0, int c = 0);      // Ok
int f2(int a,     int b = 0, int c = 0);      // Ok
int f3(int a = 0, int b = 0, int c);          // Error
int f4(int a = 0, int b,     int c = 0);      // Error
```

I/O:

הערוצים הסטנדרטיים: cin-קלט, cout-פלט, cerr-שגיאות (נמצאים ב-std namespace).

```
#include <iostream>
using std::cin;
using std::cout;
int i;
while (!(cin.eof())) {   ← כל עוד לא הגענו לסוף הקלט
        cin >> i;
        cout << "param: " << i;
}
```

Classes:

מאפשר להגדיר אובייקט ולייחס אליו פונקציות ואופרטורים.

בניגוד ל-struct, במחלקות משתנה מוגדר כברירת מחדל כprivate.

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

private: שימוש אפשרי ע"י member-function או friend.

protected: שימוש אפשרי ע"י member-functions או מחלקות יורשות.

©ענת עציון & עדי פוקס

המצביע this:

כאשר נקראת מתודה כלשהי, נשלח לה פרמטר סמוי בשם: 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

פונקצית CONST	לא משנה את הTHIS
	עצמים שמוגדרים כconst יכולים להפעיל רק את הפונקציות האלה כי הן מתחייבות שאינן משנות אותם.
		Class String
		{
		public:
		int length() const;
		}

פונקצית STATIC	אין לה THIS ניתן לקרוא לה ישירות מתוך הSCOPE של הCLASS. פונקציה כזו היא למעשה שרות שניתן לחשוב על השימוש בו כפונקציה רגילה בC	Class String
		{
		public:
		static char* ToLower(char* s)
		{…}
		}
		char* s1 = String::ToLower("ABC");

פונקציות או מחלקות FRIEND
לא משתמשים בFRIEND במבחן!
 	
כלFRIEND METHOD ניתן להחליף בMETHOD גלובלי שקורא לMETHOD PUBLIC שניגשת לשדות PRIVATE
	פונקציות non member או מחלקות שמחלקה נותנת להן גישה למשתני PRIVATE שלה	Class String
		{
		friend int func1();
		private :
		int a;
		}
		int func1()
		{
		String s;
		s.a = 123;
		}

- מחלקה A friend של B לא גורר: מחלקה B היא friend של A.
- מחלקה A friend של B ומחלקה B friend של C לא גורר: מחלקה A friend של C.
- ייתכנו שתי מתודות עם אותו שם ואותם פרמטרים, אך אחת const והשנייה לא (ה-const תיקרא עבור אובייקטים מסוג const).

©ענת עציון & עדי פוקס

inline functions:

ניתן להגדיר בשתי דרכים: 1. כחלק מהגדרת המחלקה. 2. להוסיף inline לפני המימוש. בכל-מקרה על פונקציה זו להיות ממומשת ב-header file.

```
class Stack {
    int _size;
    int top_index;
    int* array;
public:
    int size()
        { return _size; }
};
```

```
class Stack {
    int _size, top_index;
    int* array;
public:
    int size()
};

inline
int Stack::size()
{
    return _size;
}
```

יתרונות/חסרונות:

- מגדיל מהירות (אין קריאות לפונקציה).
- מקטין מהירות (עלול לגרום לגישות רבות לזיכרון).
- מגדיל כמות הקוד(מנפח את הקוד בגלל השכפול)
- מקטין כמות הקוד (נחסך הקוד של הקריאה לפונקציה וחזרה מהמחסנית).
- לא משפיע (רוב המערכות אינן מוגבלות ע"י זמן ה-CPU).

Constractor & Destructor:

Constractor: מצב שבו עצם מאותחל בלי ערך מוגדר הוא מצב לא תקין לכן לכל מחלקה יש פונקציה בעלת שם המחלקה-constractor שמטרתה לאתחל עצם ברגע הגדרתו ואת כל ה-data-members שלו. אם לא הוגדר C'tor, הקומפיילר מקצה אוטומאטית C'tor חסר-ארגומנטים שלא עושה כלום (מומלץ לא להסתמך עליו).

Destructor: יוגדר כאשר התבצעה הקצאה דינאמית בתוך המחלקה ואז על הפונקציה לשחררו. ברוב המקרים לא קוראים לו והוא נקרא אוטומאטית כאשר ה-scope של האובייקט מת (הבלוק בו הוא מוגדר נגמר או שנזרק exception שלא נתפס על-ידיו). שמו כשם המחלקה עם ~ לפני.

- C'tor ו-D'tor לא מחזירים ערכים.

זמני קריאה:

אופן הקצאת אובייקט	C'tor	D'tor
משתנים לוקאליים	כל פעם שהתוכנית מגיעה להכרזת המשתנה	כל פעם שהתוכנית יוצאת מהתחום בו הוגדר המשתנה.
משתנים דינאמיים	כל פעם שמוקצה אובייקט ע"י new	כל פעם שאובייקט משוחרר ע"י delete
משתנים גלובליים	עם תחילת התוכנית (לפני ה-main)	עם סיום התוכנית (לאחר סיום main)
משתנים סטאטיים	בפעם הראשונה שהתוכנית מגיעה לתחום בו מוגדר המשתנה	עם סיום התוכנית

©ענת עציון & עדי פוקס

תכונות נוספות:

- כדי להקצות מערך של אובייקטים, למחלקה שלהם צריך להיות **C'tor חסר ארגומנטים** (או שיש לו **ערכי default לכל הפרמטרים** כך שאינו חייב לקבל פרמטרים במפורש).
- ניתן להגדיר מחלקות אשר יש בהן שדות שהם עצמם עצמים של מחלקות אחרות.
  סדר יצירה: 1. members 2. אובייקט עצמו.
  סדר הריסה: 1. אובייקט עצמו. 2. members.
- ניתן לקרוא ל-C'tor מפורשות, הדברים הבאים שקולים:

```
String one("abc");   ← the C'tor is applied to one
String one = String("abc");   ← the C'tor creates a temporary object, which is copied to one
```

- מרגע שמומש **איזשהו** C'tor, ה-C'tor חסר הארגומנטים לא קיים יותר.

רשימות אתחול:

- הדרך המקובלת לאתחל שדות פנימיים של מחלקה. כדאית ע"פ "אתחול" בתוך הפונקציה כיוון שחוסכים אתחול ראשוני בזבל.
- סדר הפעלת היוצרים אינו הסדר בו הם מופיעים ברשימת האתחול אלא הסדר בו השדות מופיעים בהגדרת המחלקה.

```
Stack::Stack(int s) : size(s) {}   ← רשימת אתחול
```

- אסור לאתחל משתנים סטטיים ברשימת האתחול.

כללים מסכמים:

- ניתן לשים ערכי default לC'tor כדי לאפשר לcompiler חופש לקריאה גם אם לא הועברו כל הארגומנטים.

```
Stack (int size = 0): (רשימת אתחול){};
```

- C'tor טוב הוא C'tor ריק מאחר וכל הmembers של הclass נוצרים בשלב רשימת האתחול (מלבד משתנים סטטיים) – לכן מאתחלים אותם בשלב רשימת האתחול. לדוגמא:

```
Stack (int size): <= כאן למעשה נוצרים כל ה data members לכן כאן תהיה רשימת האתחול => {};
```

- ברשימת אתחול של מחלקה יורשת (Derived) **חייב להופיע C'tor של המחלקה הנורשת (Base)**
- **C'tor** אף פעם לא יכול להיות virtual, D'tor יכול להיות!
- במידת האפשר, ה-D'tor לא יכול לזרוק execpetions כי אין מה לעשות איתם.

Operator Overloading:

הגדרה:

```
<return type> operator<operator name> (<arguments>);
```

1. פונקציה חיצונית.
2. method של המחלקה (ואז אין צורך להעביר את האובייקט של המחלקה-this).

©ענת עציון & עדי פוקס

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 this)

מגבלות:

- ניתן להעמיס רק אופרטורים שכבר קיימים.
- האופרטורים מקבלים את אותו מספר משתנים
- אותו סדר עדיפות ואותה אסוציאטיביות.

ערכי החזרה:

- איבר זמני- כאשר הפעולה יוצרת איבר חדש אותו רוצים לקבל. למשל: a+b.
- רפרנס- כאשר רוצים לקבל את האובייקט לאחר השינוי. למשל: 3=+(2=+a), ואז ניתן לשרשר פעולות.

++a תחזיר איבר זמני והפעולה a++ תחזיר רפרנס.

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 (גרסת Non-friend) :

- לא ניתן להגדיר כפונקצית member כי הארגומנט הראשון הוא עצם השייך למחלקה אחרת (ערוץ קלט/פלט).

©ענת עציון & עדי פוקס

- על הפונקציה המחזירה את האופרטור להחזיר כרפרנס את הערוץ כדי שיהיה אפשר להמשיך להזרים לו נתונים.

```
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)};
```

Copy Constructor & Assignment Operator:

עבור כל מחלקה חדשה מוגדרות באופן אוטומאטי שתי פונקציות:

1. **copy constructor:**
   a. אתחול של עצם אחד מהמחלקה ע"י עצם אחר מאותה מחלקה.
   b. אתחול פרמטרים בקריאה לפונקציה– כאשר מעבירים פרמטר by value.
   c. החזרת ערך מפונקציה by value.
2. **אופרטור ההשמה (=):**
   a. השמה של עצמים מהמחלקה זה לזה פרט לאתחול של עצם אחד של המחלקה (במקרה של אתחול ייקרא ה-copy C'tor גם בשימוש ב= באתחול).

מימוש אופרטור השמה:

```
Class Numbers {
private:
        int* array_holder;
        int size;
public:
        Person (int s=0): array_holder = new int[s]; size =s; {}….
}
Numbers::operator= (const Numbers & right) {
/* נמנע מהשמה עצמית כדי למנוע מצב שאחרי שחרור ננסה לשים ערך חדש על איזור ששוחרר וכבר לא שייך לתוכנית */
        If (this == &right) return *this
        Delete [] array_holder;   // שחרור משאבים
        array_holder = new array [right.size]  // הקצאת משאבים חדשים
        (COPY ARRAYS…)  // העתקת שדות
        return *this;  // returning this
}
```


©ענת עציון & עדי פוקס

מימוש Copy C'tor:

```
Stack::Stack(const Stack& st) {
        array = new int[st.size];
        if (array == 0)     //assums error exists
                error("out of memory");
        size = st.size;
        top_index = st.top_index;
        for (int top = 0; top < st.top_index; top++)
                array[top] = st.array[top];
}
```

- לא צריך לבדוק השמה עצמית (אין לכך משמעות).

כללי אצבע:

- מומלץ לשתף קוד בין אופרטור השמה ו-C'tor copy ע"י פונקציות פרטיות.
- **The BIG Three:** אם הגדרנו מחדש את אחד מהשלושה: אופרטור השמה, D'tor, copy C'tor בד"כ נידרש להגדיר מחדש את שלושתם. ונידרש לכך כמעט תמיד במחלקה שמכילה DATA MEMBERS שמוקצים באופן דינמי.

המרות טיפוסים:

1. **המרה ע"י C'tor:**

```
T(class C);  //converts from class C to T
```

- ההמרה מתבצעת ע"י הגדרת C'tor המקבל ארגומנט יחיד, מהטיפוס הנ"ל.

```
class Complex {
        Complex(double r): re(r), im(0) {}
        friend Complex operator+(Complex,Complex);
};
Complex a, b;
a = b+23;
```

מה שמתבצע:

1. 23 מומר ל-Complex ע"י ה-C'tor.
2. מתבצע חיבור בין שני Complex.
3. a מקבל את התוצאה ע"י אופרטור ההשמה המוגדר כברירת-מחדל.

1. **המרה ע"י אופרטור:**

```
operator C() // converts from T to C
```

- בעזרת אופרטורים מיוחדים שמוגדרים במחלקה (member-functions) אפשר להמיר את המחלקה לטיפוסים פנימיים של השפה או למחלקות אחרות שכבר הוגדרו.
- זו הדרך היחידה להגדיר המרה לטיפוסים פנימיים של השפה.

```
class Price {
        int shekels, agorot;
public:
        operator double() {return shekels+agorot/100.0}
};
```


©ענת עציון & עדי פוקס

כללים:

- מתבצעת רק המרה אחת שנכתבה ע"י המשתמש .
- גם עבור אופרטורים יכולה להתבצע המרה.
- עדיף להגדיר אופרטורים סימטריים ע"י פונקציה חיצונית (כי לא יכולה להתבצע המרה על הארגומנט הראשון-this בפונקציה שהיא member וצריך שתהיה סימטריה בין שמאל לימין).

תכנות גנרי – Templates:

Function Templates:

עבור פונקציה שמבצעת את אותו תהליך עבור מספר שונה של פרמטרים או סוגים שונים של פרמטרים ניתן היה להגדיר Overloading לפונקציה. רמה גבוהה יותר ניתן להשיג במידה ויש מספר קבוע של משתנים מטיפוס מסוים ואנחנו רוצים לחזור על אותה פעולה עבור כל טיפוס באופן הבא:

```
template<class T>

void Swap(T& i, T& j) {
        T temp = j;
        j = i;
        i = temp;
}
```

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.

- הקומפיילר רואה כי זו תבנית ולא פונקציה אמיתית ולכן לא מקמפל את הקוד, אלא מייצר חורים.
- המימוש מזכיר מאקרו חכם ולכן כל הקוד של ה-template נמצא ב-header file או כ- inline function.
- לפעמים ה-template מניח הנחות לגבי קיומן של פונקציות/מתודות מסוימות בטיפוס.
- ניתן להעביר ל- template פרמטר שאינו טיפוס למשל: מחרוזת, שם פונקציה או קבוע.

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.

*הקומפיילר לא מאפשר המרות של טיפוסים על תבניות.

©ענת עציון & עדי פוקס

Class templates:

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

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

העמסת אופרטור [] – const correctness:

צריך שתי גרסאות: הגרסה הנכונה תיקרא בהתאם לעצם. וכך עצם שהינו const גם האובייקטים המוכלים בו יהפכו ל-const.

```
const T& operator[] (int index) const;

T& operator[](int index);
```


©ענת עציון & עדי פוקס

הורשה:

הבעת יחסי is-a בתוכנית. משמשת לשתי מטרות שונות:

1. **code reuse:** כאשר נרצה כי מספר מחלקות יהיו בעלות התנהגות זהה, כולן תירשנה ממחלקת-אב משותפת אשר תממש התנהגות זו. בדרך זו נמנע משכפול קוד.
2. **polymorphic behavior:** כאשר נרצה כי מספר מחלקות יחשפו ממשק זהה אך יתנהגו בצורה שונה.

כללים:

- Private- עוברים בירושה אבל **לא ניתן** לגשת ל private של מחלקת-האב מתוך החלקה היורשת.
- Protected- עוברים בירושה ו**ניתן לגשת** ל protected של מחלקת-האב מתוך החלקה היורשת.
- אתחול: בכל פעם שיוצרים אובייקט בן, בעצם יוצרים גם אובייקט מסוג האב.
  - ברשימת האתחול של המחלקה היורשת נקרא הC'tor של מחלקת-האב כלומר חייבים להצהיר עליו (או להגדיר C'tor, חסר ארגומנטים למחלקת-האב).

```
class MultiStack: public Stack {
public:
    MultiStack (int size) : Stack (size) {}
```

הקונסטרקטור של מחלקת-האב ←

  - באתחול של מחלקה יורשת נקרא הC'tor של מחלקת האב (לאתחול השדות של האב) **לפני** יצירת שדה כלשהו של המחלקה היורשת (ובפרט לפני שנקרא ה-C'tor של הבן).
  - הריסת מחלקת האב נעשית **אחרי** הריסת המחלקה היורשת (בפרט אחרי ה-D'tor שלה).

סדר בנייה:
1. בנאי מחלקת האב
2. בנאי השדות, לפי סדר ההצהרה עליהם במחלקה.
3. גוף הבנאי של המחלקה

הריסה: בסדר הפוך

- קונפליקט שמות:
  - הגדרת מתודה עם שם זהה למתודה במחלקת-האב "תבטל" (Override) את המתודה של האב במחלקה היורשת.
  - ניתן לגשת ל-member function של האב למרות שנכתבה פונקציה בשם זהה במחלקת הבן, ע"י ציון שמה המלא:

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

שימוש בפונקציה ממחלקת-האב ←

- Constructors:
  - C'tor, Copy C'tor, D'tor ואופרטור השמה אינם עוברים בירושה!
  - אם אין למחלקה היורשת Copy C'tor ומעתיקים לתוכה אובייקט מסוג מחלקת-האב, ייקרא ה Copy C'tor של מחלקת-האב.
  - אם אנחנו קוראים ל-Copy C'tor של המחלקה היורשת עם המחלקה היורשת לא נקרא שום C'tor של ממחלקת האב.
  - אם אנחנו קוראים לC'tor של מחלקה יורשת עם עצם מסוג מחלקת האב ייקרא הCopy C'tor של מחלקת האב.

©ענת עציון & עדי פוקס

פולימורפיזם:

אם אנו משתמשים במצביע לעצם מסוים ואנחנו רוצים לקרוא לmethod שלו, אז אם המצביע הוא מסוג מצביע של מחלקת האב אז תיקרא הmethod של מחלקת האב.

פונקציה וירטואלית:

האופן המדויק בו יתנהג האובייקט תלוי בטיפוס הבן. למי שנעזר בטיפוס האב לא משנה מה הטיפוס המדויק של הבן והא יכול לעבוד עם כולם באופן אחיד.

```
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
```

- טוב עבור מערך מצביעים לאובייקטים של-base שמעבר עליו ייקרא לפונקציה הספציפית של כל derived.

C'tor, D'tor & Virtual:

- C'tor שקורא ל-Virtual לא יכול לקרוא לגרסה של מחלקת-הבן. ולכן נקרא ה-virtual של מחלקת האב. ולכן C'tor לא יהיו virtual אף-פעם.
- בכל מחלקה שיש בה פונקציה virtual או אפילו אם יש לה יורשים שלהם פונקציות virtual, חייבים להגדיר virtual D'tor.

Abstact classes-ו Pure Virtual

לעיתים אנו נרצה ליצור מחלקה מופשטת - זו מחלקה שלא ניתן להשתמש בה אלא רק לרשת ממנה. זוהי מחלקה עם פונקציה pure virtual (אחת או יותר). לא ניתן ליצור אובייקטים של מחלקה זו.

```
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 */
```


©ענת עציון & עדי פוקס

Execeptions:

בעיות:

- פונקציה שאינה מחזירה ערך (void), צריכה להודיע על שגיאה.
- מרחק גדול בין הפונקציה שזיהתה את השגיאה לזו שיודעת לטפל בה.

הרעיון: לאפשר העברת ערך השגיאה לרמה המתאימה שיכולה לטפל בשגיאה, תוך דילוג על שלבי-הביניים. בכל-פעם שנרצה להודיע על שגיאה "נזרוק" אובייקט שמייצג אותה. זריקה זו תגרום לעלייה במעלה היררכיית הקריאה לפונקציות (תוך שחרור משתנים מקומיים בעזרת ה-D'tor שלהם) עד לבלוק ה-try-catch הראשון המתייחס לשגיאה זו.

try-catch-throw:

```
try {
  // code here
}
catch (int param)   { cout << "int exception"; }
catch (char param)  { cout << "char exception"; }
catch (...)         { cout << "default exception"; }
```

try- הגדרת בלוק בו ייתכנו Exceptions.

catch- תפיסת exception.

throw- זריקת exception.

בהצהרה על פונקציה ניתן להגביל את סוגי ה-exceptions שהיא תזרוק. ואם לא מגדירים אף סוג (נשאר ריק) אז הפונקציה לא מורשית לזרוק שום-דבר. פונקציות אחרות יורשו לזרוק כל-דבר.

```
float myfunction (char param) throw(int);   // only int exceptions
int   myfunction (int param)  throw();      // no exceptions allowed
int   myfunction (int param);               // all exceptions allowed
```

תפיסה וזריקה מייד: ניתן לתפוס exception, לעשות משהו בעקבותיו ואז להמשיך לזרוק אותו:

```
catch (std::bad_alloc& e) {     throw בלי טיפוס- לזרוק את מי שנתפס, כאשר לא בטוחים
                                מהו הטיפוס אך רוצים כי הוא ימשיך למעלה <-
      cerr << e.what(); throw;
}
```

הדרך המומלצת למימוש:

הגדרת class של שגיאות שיורש מ std::exception וכל השגיאות יורשות ממנו. המחלקה exception מבטיחה שלכל מה שיורש ממנה יש מתודה ()what שמחזירה מחרוזת שאפשר להדפיס:

```
class str_err: public
                std::exception {};
class empty : public str_err {};
class dot   : public str_err {};
class bad_ch: public str_err {};
```

```
} catch(const empty& e) {
  cerr << e.what();
```

בנוסף, ניתן לוותר על (…) catch כי כל ה-exception-ים יורשים מstd::exception. ולכן בסוף יהיה: catch (exception& e) למי שלא התכוננו אליו.

כללים:

- לא ניתן שיהיו שני exception ביחד ולכן D'tor לא זורקים exception.
- אם C'tor זורק exception (זה כן מותר) אז ה-D'tor של האובייקט לא נקרא.
- נקראים D'tor רק למשתנים מקומיים ולא לזיכרון שהוקצה דינאמית.
- כשיש טיפוסי שגיאות שונים אז ניתן לעשות catch במקומות שונים בתוכנית (מומלץ).

©ענת עציון & עדי פוקס

The Standard Template Library:

מכילה containers ואלגוריתמים , מאפשרת הרחבה ע"י המשתמש ושומרת על ביצועים גבוהים.

Standard Library container class	Description
Sequence containers	
vector	fast insertions and deletions at back direct access to any element
deque	fast insertions and deletions at front or back direct access to any element
list	doubly linked list, fast insertion and deletion anywhere
Associative containers	
set	fast lookup, no duplicates
multiset	fast lookup, duplicates allowed
map	one-to-one mapping, no duplicates, fast key-based lookup
multimap	one-to-many mapping, duplicates allowed, rapid key-based lookup
Other container	
stack	last-in, first-out (LIFO)
queue	first-in, first-out (FIFO)

איטרטורים:

לא מחלקה, אלא מושג (concept). משמש למעבר סדור על איברי ה-container.

begin()- מחזירה איטרטור לאיבר הראשון באוסף.

end()- מחזירה איטרטור לאיבר "דמה" אחרי האיבר האחרון.

אלגוריתמים גנריים:

על הפונקציה לפעול על כל container המכיל כל סוג אובייקט:

```
template<typename Iterator>
Iterator max(Iterator start, Iterator end) {
  if (start == end) {
      return end;
  }
  Iterator maximum = start;
  for(Iterator i = ++start; i != end; ++i) {
      if (*i > *maximum) {
            maximum = i;
      }
  }
  return maximum;
}
```

Function Objects:

אובייקט המעמיס את האופרטור () (אופרטור ההפעלה).

```
class LessThan {
public:
        LessThan (int n) : n(n) {}
        bool operator() (int m) {return m < n; }
private:
        int n;
};
```

המחלקה מוגדרת כפונקציה המקבלת int ומחזירה bool

©ענת עציון & עדי פוקס

מצביעים חכמים:

בעיה: מצביעים אחראים לרובן המוחלט של שגיאות הזיכרון. ירושות דורשות שימוש במצביעים.

פיתרון: יצירת מחלקה המתנהגת כמצביע אך מוסיפה התנהגות נוספת לשימוש בטוח יותר.

- שיפור: מצביע משותף: לכל עצם מוצבע יישמר גם מספר ההצבעות שלו כדי לדעת מתי יש לשחרר גם את העצם בנוסף לשחרור המצביע.

המרות-casting:

המרה ב-C: (type) expression

תוחלף ב++C ל: castType<type>(expression)

כאשר castType הינו אחד מ:

static_cast:

- פועל בזמן קומפילציה
- מתאים לביצוע המרות מפורשות הנתמכות ב-C'tor או אופרטורי המרה שהוגדרו במחלקות.
- מתאים ל-downcast (המרה במורד עץ ההורשה- ממחלקת אב למחלקת-בן)

dynamic_cast:

- פועל בזמן ריצה.
- נועד להמיר פויינטר/רפרנס למחלקת אב לפויינטר/רפרנס למחלקת הבן.
- מתבצעת בדיקה בזמן ריצה של חוקיות ההמרה.
- ניתן להפעיל רק על פוינטרים/רפרנסים של מחלקות שיש להן לפחות virtual אחד.

const_cast:

- מיועד להסרת const מאובייקט.
- משמש בעיקר להעברת ארגומנט const לפונקציה שמקבלת פרמטר שאינו const (כאשר לא ניתן לשנות את הפונקציה, נכתבה ע"י גורם אחר...).

reinterpret_cast:

- משמש לביצוע המרות שאינן מוגדרות במפורש (אחרת נבצע עם static_cast).
- המרות מפויינטר כלשהו ל-int או להפך.

©ענת עציון & עדי פוקס

Template לקובץ ממשק:

MYCLASS.H

```
#ifndef MYCLASS_H
#define MYCLASS_H


class MyClass {
public:
	static const int MAX_PARAM;
```

קונסטרקטורים מסוגים שונים:

```
	// C'tor - ALWAYS PREFERRED WITH DEFAULT PARAM
	MyClass(const T& val = T(), int size = 0, int param);

	// C'tor for array - ALWAYS PREFERRED WITH DEFAULT PARAM
	MyClass(const T* val = NULL, int size = 0);

	// D'tor
	~MyClass();
```

קופי-קונסטרקטור:

```
	// Copy C'tor
	MyClass(const MyClass& other);
```

אופרטור השמה:

```
	// assignment operator
	MyClass& operator= (const MyClass& other);
```

אופרטורים היוצרים אובייקט חדש- מחזירים איבר זמני מסוג המחלקה ואינם משנים את האובייקטים עליהם הם פועלים:

```
	// operator +
	MyClass operator+(const MyClass& right) const;

	// operator /
	MyClass operator/(const MyClass& right) const;
```

אופרטורים המשנים אובייקט ומחזירים רפרנס אליו לאחר השינוי (על-מנת שנוכל לשרשר פעולות):

```
	// operator +=
	Myclass& operator+=(const MyClass& right);

	// operator *=
	MyClass& operator*=(const MyClass& right);
```

אופרטורי השוואה, מחזירים ערך בוליאני ולא משנים את האובייקטים:

יכול להיות גם פונקציות גלובליות וגם member

```
	// operator ==
	bool operator==(const MyClass& other) const;

	// operator <
	bool operator<(const MyClass& other) const;
```


©ענת עציון & עדי פוקס

אופרטורי קלט ופלט (לא יכולים להיות member) לא להגדיר כ-friend:

```
// output operator
ostream& operator<<(ostream& os, const MyClass& my_class);

// input operator
istream& operator>>(istream& is, MyClass& my_class);
```

כאשר נדרש טיפול בשגיאות, להוסיף בסוף הממשק:

```
class MyClassException: public exception {};

class objectNotFounf: public MyClassException {};
class objectAlreadyInMyClass: public MyClassException {};
class MyClassFull: public MyClassException {};


#endif
```

לשים-♥:

- עדיף לא לעשות פונקציית set לשדה, אלא כמה פונקציות כמשמעות מה שרוצים לשנות, למשל פונקציית open ופונקציית close
- לא לשכוח const על כל אובייקט שהפונקציה לא משנה.
- שדות ב-private לא צריכים להיות const.
- לבדוק האם כאשר מאתחלים עם string זה צריך להיות סתם string או &const string.
- לשים-לב האם נאמר במפורש באיזה טיפוס נתונים יחזיק ה-Class את האובייקטים שלו.

©ענת עציון & עדי פוקס

Template להורשה ופולימורפיזם:

```
// see comments when
virtual ~Base() {}

// defining "is-a"
class Son: public Base{
public:
```

וירטואלית שמחזירה כתובת (מצביע)

```
// returning the address of a new cloned object
// (no dynamic memory allocs)
(base.h)     virtual Base* clone() const = 0;
(son.h)      virtual Base* clone() const { return new Son(*this); }
```

וירטואלית שמדפיסה למסך:

```
// can be used when overloading operator<< (output)
(base.h)     vitual void print(ostream& os) const = 0;
(son.h)      void print(ostream& os) const;
(son.cpp)    void Son::print(ostream& os) const {
                   os << param << endl;
             }
```

וירטואלית שמחזירה ערך:

```
// the example of shapes from the tutorial
(base.h)     virtual double func() const = 0;
(son.h)      double func() const { return (param1*param2); }
```

לשים-♥:

- לשים ב-protected את השדות הפנימיים של מחלקת הבסיס.
- עושים virtual D'tor במקרים הבאים:
  - יש הקצאה דינאמית ולכן יהיה שימוש ב-delete
  - נוצר מצביע למחלקת הבסיס שבעצם מצביע לבן ובסוף הבלוק צריך להיקרא ה-D'tor הנכון (גם אם זו אינה מחלקה אבסטרקטית!)
  - כאשר למחלקה יש vitual method לא בהכרח יש virtual D'tor!
    - כאשר כל מצביע לאובייקט הינו מסוגו ולא מסוג מחלקת הבסיס
- אופרטורים שמומשו במחלקת הבסיס כן עוברים בירושה למחלקה שיורשת.

©ענת עציון & עדי פוקס

Template לשימוש בתבניות גנריות:

```
template <class T>
class Array : public ArrayBase {
private:
	T* elements;
```

פונקציה פנימית המשמשת את שני סוגי ה-copy C'tor:

```
	void fillArray(T* data, int sz) {
		elements = new T[sz];
		size = sz;
		for (int i=0; i<sz; i++)
			elements[i] = data[i];
	}

public:
```

קונסטרקטור למערך:

```
	// usage: Array<int> a(3);
	Array(int sz): ArrayBase(sz), element(new T[sz]) {
	}
```

קופי-קונסטרקטור ממערך-Array מאותו הטיפוס, בלבד:

```
	// usage:
	Array(const Array<T>& array2) : ArrayBase(array2.size) {
		fillArray(array2.elements, size);
	}
```

קופי-קונסטרקטור ממערך רגיל מאותו הטיפוס:

```
	// usage: Array<int> = a(regular_int_array, 4);
	Array(T* array2, int sz) : ArrayBase(sz) {
		fillArray(array2, size);
	}
```

אופרטורים עבור מערכים:

```
	// set. For example - T[index] = 77;
	T& operator[](int  index);

	// get. For example -  int i = T[index];
	const T& operator[](int index) const;
```

דיסטרקטור עם [] מיועד למערך:

```
	~Array() { delete[] elements; }
```

אופרטור השמה:

```
	Array& operator=(const Array<T>& other) {
		if (this == &other)
			return *this;
		delete[] elements;
		fillArray(other.elements, other.size());
		return *this;
	}
};
```

מימוש קונסטרקטור בירושה:

```
template<class T>
Array<T>::Array(int size): ArrayBase<T>(size) : {
	for (int i=0; i<size; i++){
		…
	}
}
```


©ענת עציון & עדי פוקס

אופרטורי קלט ופלט (פונקציות גלובליות):

```
template<class T>
ostream& operator<<(ostream& out, const Array<T>& arr) {
	for (int i=0; i<arr.getSize(); i++)
		return out << arr[i] << ' ' << endl;
}

template<class T>
istream& operator>>(istream& in, Array<T>& arr) {
	for (int i=0; i<arr.getSize(); i++)
		in >> array[i];
	return in;
}
```

טמפלייט מסוג יותר ספיציפי שיש לו גם פרמטר של גודל (int):

```
template<class T, int SZ>
class ArraySize: public Array<T> {
public:
	// usage: ArraySize<int,3> a;
	ArraySize(): Array<T>(SZ) {
	}
};
```

קונסטרקטור חסר ארגומנטים

לשים- ♥:

- כאשר הפעולה מחזירה איבר זמני ולא משנה אובייקט קיים אז לא להחזיר רפרנס. רק כאשר משנים אובייקט קיים!
- אם נתון קוד ובו מאתחלים אובייקט שהוא- const אז לשים-לב שכל פונקצייה המופעלת עליו תהיה לה גם גרסה הפועלת על אובייקט שהוא const (ואז בד"כ גם הארגומנט וגם האובייקט יהיו const).
- בתוך המימוש לכתוב template <class T> לפני כל פונקציה.

דוגמאות להנחות לגבי הטיפוס-T:

- copy C'tor- על מנת שנוכל לממש את ה-Copy C'tor של ה-class.
- אופרטור << על-מנת שניתן יהיה לקלוט מידע מה-istream.
- אופרטור >>- כדי לממש את אופרטור ההדפסה של הclass.
- אופרטור השמה- כדי שנוכל לממש את האופרטור השמה של ה-class.
- אופרטור * ו-+ כדי שנוכל לממש את האופרטור * של ה-class.
- אופרטור +,-,*- כדי לבצע חישוב באחת הפונקציות של ה-class.
- default constructor- לאתחול מערך מהטיפוס T עבור ה-counstructors של ה-class.

©ענת עציון & עדי פוקס

דברים שונים במימוש של-Class:

MYCLASS.CPP

```
const int MyClass::MAX_PARAM = 100;
```

אתחול מערך של מצביעים בתוך ה-קונסטרקטור:
(המערך כבר נוצר כשדה פנימי ואותחל "אתחול ראשוני" לערכי ה-C'tor חסר ארגומנטים, אך כעת צריך לאתחל אותו לערכים תקינים-NULL)

```
MyClass::MyClass(string name, int num) : name(name), num(num) {
	for (int i=0; i<MAX_PARAM; ++i) {
		array[i] = NULL;
	}
}
```

שימוש בפונקציה הפוכה שכבר מומשה:

```
void MyClass::trnasferFrom(MyClass& other, const Obj& object) {
	other.transferTo(*this, object);
}
```

הבדלה בין שימוש ב-> (עבור מצביע) לבין שימוש בנקודה:

```
Car real_car;
real_car.func();
Car* car_ptr;
car_ptr->func();
```

מימוש דיסטרקטור:
עושים []delete לכל שדה שהוא מסוג *type כלשהו.

```
~Myclass() {
	delete[] name; // if private- *char name
```


©ענת עציון & עדי פוקס

דוגמאות קריאה ל- C'tors:

Numbers c1,c2;	Ctor- הכרזה על משתנה
Numbers c1 = c2;	Copy C'tor- אתחול של משתנה בעזרת משתנה אחר
c1 = c2;	Operator=- השמה של עצמים אחד לשני שאינה אתחול.
func(Numbers c1)	Copy C'tor- בקריאה לפונקציה ללא רפרנס למשתנה, מועתק ערכו אל המשתנה המקומי של הפונקציה.
func(Numbers& c1)	שום-דבר- רק שינוי מצביע (רפרנס) כך שיצביע אל משתנה.
Numbers& c2 =	שום-דבר- שינוי מצביע כך שיצביע לאובייקט (מתן שם אחר).
Numbers c1 = Numbers(12);	C'tor- אתחול משתנה.
func(Numbers(12));	C'tor- אתחול משתנה במהלך שליחה לפונקציה.

Numbers func()
{
… return Numbers(12);
}
	C'tor- אתחול משתנה המוחזר מפונקציה.

Numbers func()
{
Numbers c1(15); /*ctor*/
return c1; /*copy ctor*/
}
	C'tor- לאתחול המשתנה
	Copy C'tor- להעתקת ערך המשתנה לפרמטר שחוזר מהפונקציה.

Numbers& func()
{
Numbers c1(15); /*ctor*/
return *c1; /*error (either syntax or run time – c1 dies at the end of the function*/
}
	C'tor- אתחול משתנה
	טעות- החזרת כתובתו של איבר זמני הנהרס בסיום הפונקציה.

Number (1.7 + c2)	C'tor- ליצירת האובייקט החדש Copy C'tor- להשמת ערך האובייקט לפונקציה
class B: public A B* ptr_b = new B /* A C'tor + B C'tor */ A* ptr_a = new B /* A C'tor + B C'tor */ delete ptr_b; /* B D'tor + A D'tor */ delete ptr_a; /* A D'tor */	A C'tor ואחריו B C'tor כאשר עושים new B. B D'tor ואחריו A D'tor כאשר מוחקים את המצביע לטיפוס B רק A D'tor כאשר מוחקים את המצביע לטיפוס A

*לא לשכוח- בסוף בלוק נקראים כל ה-D'tor-ים!

©ענת עציון & עדי פוקס