

```

Section A
-----
#ifndef COMPANY_H_
#define COMPANY_H_

typedef struct Company_t *Company;
typedef char *Name;
typedef char *Role;
typedef char *City;
typedef int StoreCode;

typedef enum { COMPANY_OUT_OF_MEMORY,
              COMPANY_BAD_ARGUMENT,
              COMPANY_WORKER_ALREADY_EXISTS,
              COMPANY_WORKER_NOT_FOUND,
              COMPANY_NOT_FOUND,
              COMPANY_NO_MANAGER,
              COMPANY_SUCCESS } CompanyResult;

typedef enum { FALSE, TRUE } Boolean;

Company      CompanyCreate(void);          or:
CompanyResult CompanyCreate(Company*);

CompanyResult CompanyAddWorkerToStore
              (Company, Name, Role, StoreCode);
CompanyResult CompanyAddStoreToCompany
              (Company, StoreCode, City);
CompanyResult CompanyMoveWorker
              (Company, Name, StoreCode, StoreCode);
CompanyResult CompanyAnyStoresInCity
              (Company, City, Boolean*);
CompanyResult CompanyGetStoreManager
              (Company, StoreCode, Name*);
CompanyResult CompanyGetNumWorkersOfRole
              (Company, StoreCode, int*);
CompanyResult CompanyIsTheSameRole
              (Company, Name, Name, Boolean*);

void CompanyDestroy(Company);             or:
CompanyResult CompanyDestroy(Company);

#endif

Section B
-----
1. #include "company.h"
#include "set.h"

struct store {
    StoreCode code;
    City city_name;
};
typedef struct store *Store;

struct worker {
    Name name;
    Role role;
    Store user_store;
};
typedef struct worker *Worker;

struct Company_t {
    Set workers;
    Set stores;
};

char *str_dup(const char *src)
{
    char *dst = malloc((strlen(src) + 1)*sizeof(char));
    if(dst == NULL) return NULL;
    return strcpy(dst, src);
}

char *store_lbl(Element e)
{
    return "";
}

Element store_cpy(Element e)
{
    Store src = (Store)e;
    Store dst = malloc(sizeof(struct store_rec));

    if(dst == NULL) return NULL;
    if(! (dst->city_name=str_dup(src->city_name))) {
        free(dst);
        return NULL;
    }
    dst->code = src->code;
    return dst;
}

void store_fre(Element e)
{
    Store s = (Store)e;
    free(s->city_name);
    free(e);
}

Boolean store_cmp(Element e1, Element e2)
{
    Store s1 = (Store)e1;
    Store s2 = (Store)e2;
    return (s1->code == s2->code);
}

char *worker_lbl(Element e)
{
    return "";
}

Element worker_cpy(Element e)
{
    Worker src = (Worker)e;
    Worker dst = malloc(sizeof(struct worker_rec));

    if(dst == NULL) return NULL;
    if((dst->name = str_dup(src->name)) == NULL) {
        free(dst);
        return NULL;
    }
    if((dst->role = str_dup(src->role)) == NULL) {
        free(dst->name);
        free(dst);
        return NULL;
    }
    dst->user_store = src->user_store;
    return dst;
}

void worker_fre(Element e)
{
    Worker w = (Worker)e;
    free(w->name);
    free(w->role);
    free(e);
}

Boolean worker_cmp(Element e1, Element e2)
{
    Worker w1 = (Worker)e1;
    Worker w2 = (Worker)e2;
    return strcmp(w1->name, w2->name)==0;
}

Company CompanyCreate(void)
{
    Company c = malloc(sizeof(struct Company_t));

    if(c == NULL) return NULL;

    c->workers = c->stores = NULL;
    c->workers = SetCreate(worker_cmp, worker_cpy,
                          worker_lbl, worker_fre);
    c->stores = SetCreate(store_cmp, store_cpy,
                          store_lbl, store_fre);

    if(c->workers == NULL || c->stores == NULL) {
        CompanyDestroy(c);
        return NULL;
    }
    return c;
}

```

3.

```

CompanyResult CompanyAnyStoresInCity
    (Company comp, City city, Boolean *ans)
{
    Store store;

    if(comp==NULL || city==NULL || ans==NULL)
        return COMPANY_BAD_ARGUMENT;

    *ans = FALSE;
    SET_FOREACH(store, comp->stores) {
        if(strcmp(city, store->city)==0) {
            *ans = TRUE;
            break;
        }
    }
    return COMPANY_SUCCESS;
}

```

```

4. COMPANY_WORKER_ALREADY_EXISTS
   COMPANY_WORKER_NOT_FOUND
   COMPANY_NO_MANAGER

```

חורף 2002-2003 מועד ב' : ADT (35 נקודות)

Section A

```

-----
#ifndef PAL_H_
#define PAL_H_

typedef struct PAL_t *PAL;
typedef const char *Pid; /* ID for points */
typedef const char *Lid; /* ID for lines */
/* But it's OK to pass "char*" instead of Pid and Lid */

typedef double Scalar;

typedef enum { FALSE, TRUE } Boolean;

typedef enum { SUCCESS,
              FAILURE,
              ILLEGAL_PARAMETER, /* NULL sent etc. */
              POINT_ALREADY_EXISTS,
              LINE_ALREADY_EXISTS,
              POINT_DOESNT_EXIST,
              LINE_DOESNT_EXIST
            } PALResult;

PAL      PAL_Create      (void);          or:
PALResult PAL_Create      (PAL*);

PALResult PAL_AddPoint(PAL, Pid, Scalar, Scalar);
PALResult PAL_AddLine (PAL, Lid, Scalar, Scalar, Scalar);
PALResult PAL_AddLineViaPoints(PAL, Lid, Pid, Pid);
PALResult PAL_DoesLineContainPoint
    (PAL, Lid, Pid, Boolean*);
PALResult PAL_AllLinesContainingPoint
    (PAL, Scalar, Scalar, Lid**, int *n);
PALResult PAL_AllLinesContainingPalPoint
    (PAL, Pid, Lid**, int *n);
PALResult PAL_GetLineCoef
    (PAL, Lid, Scalar*, Scalar*, Scalar*);
PALResult PAL_AreLinesParallel(PAL, Lid, Lid, Boolean*);

PALResult PAL_Destroy (PAL);    or:
void      PAL_Destroy (PAL);

#endif

```

Section B

1. We use 2 Sets, one for points and one for lines. (A Graph is a gross error, since the point-line relations are not as in a graph!)

2.

```

#include "math.h"
#include "pal.h"
#include "set.h"

struct PAL_t {
    Set points;
    Set lines;
    ...
};

struct Line {
    Scalar a, b, c;
};

struct Point {
    Scalar x, y;
};

PALResult PAL_AddLine
    (PAL pal, Lid lid, Scalar a, Scalar b, Scalar c)
{
    Scalar v;
    Line line;

    if(!pal || !lid) return ILLEGAL_PARAMETER;

    if(SetIsIn(pal->lines, lid))
        return LINE_ALREADY_EXISTS;

    /* first normalize... */
    v = sqrt (a*a + b*b + c*c);
    if((a == 0.0 && b == 0.0) || v == 0.0)
        return NOT_A_LINE;

    line.a = a/v;
    line.b = b/v;
    line.c = c/v;

    if(SetAdd(pal->lines, lid, &line) != SUCCESS)
        return FAILURE;

    return SUCCESS;
}

PALResult PAL_DoesLineContainPoint
    (PAL pal, Lid lid, Pid pid, Boolean *ans)
{
    Line *line;
    Point *point;
    Scalar val;

    If(!pal || !lid || !pid) return ILLEGAL_PARAMETER;

    line = SetGetData(pal->lines, lid);
    if(!line) return LINE_DOESNT_EXIST;

    point = SetGetData (pal->points, pid);
    if(!point) return POINT_DOESNT_EXIST;

    val = line->a*point->x+line->b*point->y+line->c;
    *ans = (val == 0.0 ? TRUE : FALSE);
    return SUCCESS;
}

```

3.

```

POINT_ALREADY_EXISTS
LINE_ALREADY_EXISTS
POINT_DOESNT_EXIST
LINE_DOESNT_EXIST
NOT_A_LINE (e.g., if all its coefficients are 0)

```