הטכניון - מכון טכנולוגי לישראל **מבוא למערכות תוכנה 044101**

הפקולטה להנדסת חשמל **16.2.16**

מבחן סוף סמסטר

מועד א'

משך הבחינה: שלוש שעות

נא לרשום את השם ומספר תעודת הזהות במקום המיועד במחברת הבחינה.

עליכם לענות על כל השאלות **אך ורק במחברת הבחינה**.

יש להגיש רק את מחברת הבחינה (אני אינני מחברת הבחינה) !

במבחן זה 12 עמודים (לא כולל עמוד זה) ו-4 שאלות.
שימו לב: בשאלות הבחינה מופיע קוד רב, ואתם צריכים להוסיף רק את השורות החסרות. לכן הבחינה אמנם ארוכה, אבל התשובות קצרות...

מומלץ לקרוא כל שאלה בעיון רב לפני שניגשים לפתור אותה.

השאלות בבחינה מסודרות לפי נושאי הלימוד ולוא דווקא לפי רמת הקושי !

כל חומר עזר מותר.

בהצלחה!!!

שאלה 1 - ADT (38 נק')

בכל מקום בשאלה בו יש להשלים קוד אין צורך להעתיק את הקוד הנתון. אבל, כיתבו כך שיהיה ברור איפה נכנס הקוד שכתבתם.

חלק א:

בחלק זה נבנה מערך אדפטיבי **כללי**. מערך אדפטיבי כללי הינו מערך אשר משנה את גודלו כך שכל פנייה לאינדקס אי שלילי (אפס ומעלה) היא תקינה. כאשר פונים לאינדקס בו לא הושם ערך, אז מוחזר NULL.

להלן תיאור אופן הפעולה של חמש המתודות של המערך האדפטיבי

CreateAdptArray	מאתחלת מערך ריק (כלומר ללא איברים)
DeleteAdptArray	משחררת את הזיכרון של האובייקט (כולל איבריו)
SetAdptArrayAt	מקבלת אינדקס ואיבר ושומרת **עותק** של האיבר במקום המבוקש. משחררת את האיבר הישן אם קיים.
GetAdptArrayAt	מקבלת אינדקס ומחזירה **עותק** של האיבר במיקום זה
GetAdptArraySize	מחזירה את גודל המערך (1- כאשר המערך לא אותחל בהצלחה)

בכל מקרה של כישלון יש להחזיר FAIL **או** NULL**, בהתאם למצב.**

להלן קובץ המנשק של המערך האדפטיבי.

```
typedef struct AdptArray_* PAdptArray;
typedef enum Result {FAIL = 0, SUCCESS};
typedef void* PElement;

typedef void(*DEL_FUNC)(PElement);
typedef PElement(*COPY_FUNC)(PElement);

PAdptArray CreateAdptArray(COPY_FUNC, DEL_FUNC);
void DeleteAdptArray(PAdptArray);
Result SetAdptArrayAt(PAdptArray, int, PElement);
PElement GetAdptArrayAt(PAdptArray, int);
int GetAdptArraySize(PAdptArray);
```

בחרנו לממש את המבנה ע"י מערך בגודל משתנה. כלומר נשמור מערך של מצביעים כלליים ובכל פעם שנקבל הצבה לאינדקס גדול יותר מהמערך שלנו אז נעתיק את המערך לזיכרון גדול מספיק כך שפנייה זאת תהיה חוקית. נצטרך כמובן לשמור את גודל המערך הנוכחי במבנה ובנוסף מצביעים לפונקציות המשתמש להעתקה ומחיקה של איברים.

בעמודים הבאים נתון לכם מימוש חלקי של המבנה. כאשר תתבקשו לממש אז יש לכתוב מאפס וכאשר תתבקשו להשלים יש להוסיף קוד במקומות החסרים.

א. (6 נק') השלימו את המימוש של struct AdptArray_ וממשו את CreateAdptArray.

ב. (6 נק') השלימו את המימוש של SetAdptArrayAt.

ג. (6 נק') ממשו את הפונקציה DeleteAdptArray.

```
typedef struct AdptArray_
{
        int ArrSize;
        PElement* pElemArr;
        …
}AdptArray;
Result SetAdptArrayAt(PAdptArray pArr, int idx, PElement pNewElem)
```

```
{
        PElement* newpElemArr;
        if (pArr == NULL)
                return FAIL;

        if (idx >= pArr->ArrSize)
        {
                // Extend Array
                if ((newpElemArr = (PElement*)malloc((idx + 1), sizeof(PElement))) == NULL)
                        return FAIL;
                // Init new array and copy old array to new array
(1)             …


                // Free old array and save new array
                free(pArr->pElemArr);
                pArr->pElemArr = newpElemArr;
        }

        // Delete Previous Elem and Set New Elem
(2)     …


        // Update Array Size
        pArr->ArrSize = (idx >= pArr->ArrSize) ? (idx + 1) : pArr->ArrSize;
        return SUCCESS;
}
```

חלק ב: (אין תלות בתשובות של חלק א')

כעת נשתמש במערך האדפטיבי כדי לבנות עץ כללי לפי עקרונות ה-ADT. נתייחס כאן לעץ עם שורש. כלומר, עץ הוא גרף מכוון וחסר מעגלים בעל צומת שורש אשר ממנו יש מסלול **יחיד** לכל צומת אחר. דרך ברורה יותר לנסח דרישות אלו עבור צורכי השאלה היא שכל צומת בעץ מקיים את הכללים הבאים:

1. יש לו אב יחיד (חוץ מלשורש שכמובן אין אב).
2. יש לו מספר לא מוגבל של בנים.

נבחר לצורך המימוש במבנה בו כל צומת הוא עץ בפני עצמו. בחירה זאת תאפשר להשתמש במבנה הרקורסיבי של עצים לצורך המימוש שלנו (וגם לצורך שימוש במבנה). נשתמש במערך האדפטיבי מחלק א' כדי לשמור את הבנים של כל צומת (כל בן הוא תת עץ בפני עצמו). להלן תיאור של מבנה הנתונים שלנו:

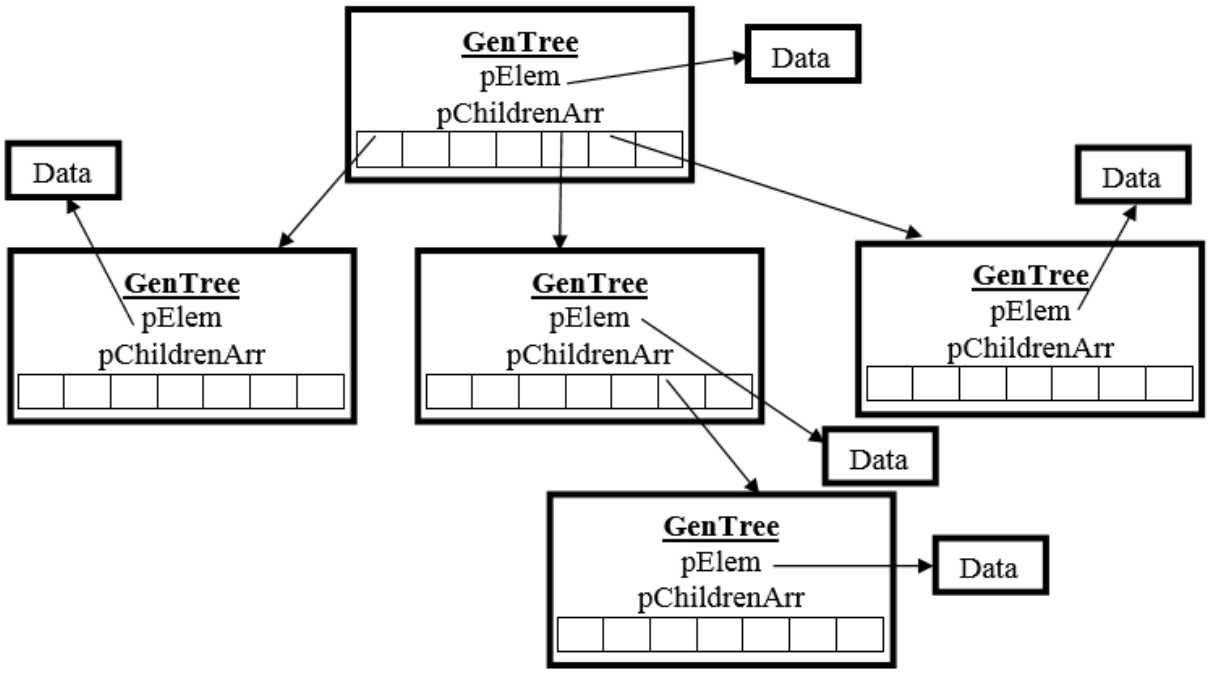
GenTree

fatherChildIdx – אינדקס העץ במערך הבנים של האב

pFather – מצביע לאב של העץ

pElem – מצביע למידע של השורש

למעשה אנו ממשים שורש של עץ ונותנים לו את היכולת לקבל שורשים אחרים בתור ילדיו. להלן דוגמא איך עץ כזה נראה (חסרים כמובן מצביעים של כל בן לאב המתאים לו): **גודל מערך הבנים לא קבוע אלא רק לצורך הציור**



מטרתנו בשאלה זאת תהיה לממש חלק מהמתודות של מבנה זה. נתונה רשימה של מתודות אלו.
כמובן שאלו לא כל המתודות של המבנה אך אלו הן העיקריות שבו. להלן הדרישות מכל מתודה:

CreateGenTree	מאתחלת שורש עץ חדש (בהמשך יכול להפוך לבן של שורש אחר). המידע שנשמר בשורש הוא **העתק** של המידע המתקבל
SetChildTreeAt	מוסיפה את העץ pChild כתת עץ ל pRoot במיקום הילד childIdx (אין העתקת מידע). אם קיים כבר תת עץ במיקום זה אז הוא נמחק. אם pChild כבר תת עץ של שורש אחר אז יש כשלון.
GetChildTreeAt	מחזירה את תת העץ (השורש שלו) במיקום הילד childIdx (אין העתקת מידע).
DeleteGenTree	מוחקת את העץ pRoot (כולל את מחיקת כל ילדיו ונכדיו וכו'...). במידה ויש לו אב אז צריך לעדכן את מערך הבנים של האב כך שלא יצביע לזיכרון לא קיים.
GetChildrenArrSize	מחזיר את גודל מערך הבנים. כלומר, את אינדקס הבן הכי גדול פלוס אחד.

בכל מקרה של כישלון יש להחזיר FAIL **או** NULL**, בהתאם למצב.**

בעמודים הבאים נתון לכם מימוש חלקי של מתודות המבנה. יש להשלים את הקוד במקומות החסרים.

ד. (4 נק') ממשו את CreateGenTree. יש **להצהיר** על כל פונקציה שדרושה לצורך השלמת המימוש
ה. (6 נק') ממשו את SetChildTreeAt. יש **לממש** כל פונקציה שדרושה לצורך מימוש זה.
ו. (5 נק') ממשו את DeleteGenTree.
ז. (5 נק') ממשו את הפונקציות שהצהרתם עליהם בסעיף ד'.

רמז: ניתן להשתמש לצורך המימוש בשאר המתודות של המבנה ללא צורך לממש אותן

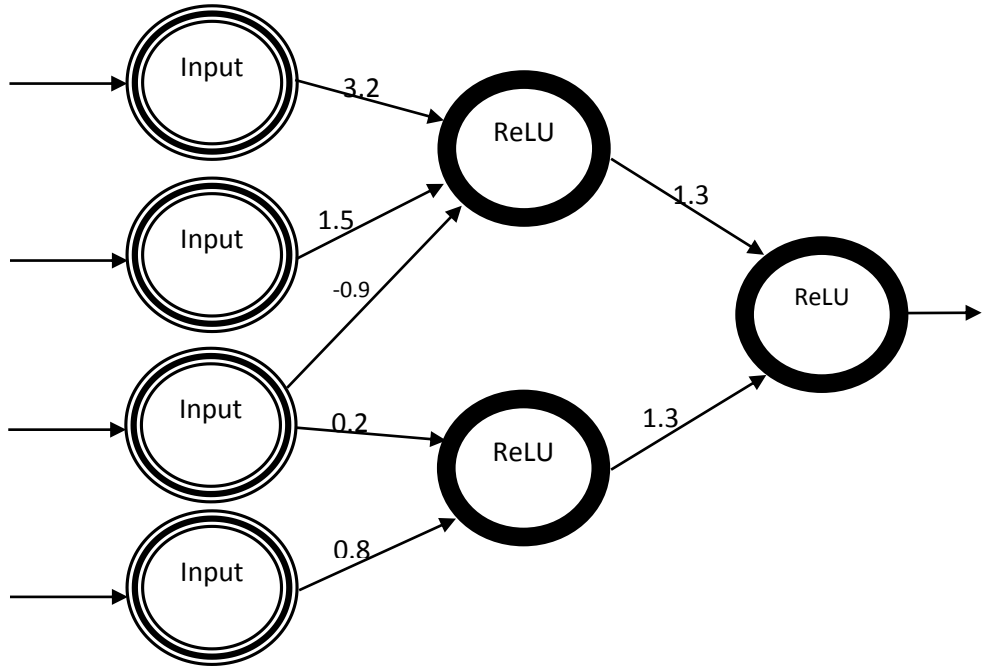
```
// Internal Function Declarations
…

// Implement Struct
typedef struct GenTree_
 {
        int fatherChildIdx;
        PGenTree pFather;
        PElement pElem;

        PAdptArray pChildrenArr;
        DEL_FUNC delFunc;
        COPY_FUNC copyFunc;
}GenTree, * PGenTree;

// Implement Methods
PGenTree CreateGenTree(PElement pElem_, COPY_FUNC copyFunc_, DEL_FUNC delFunc_)
{
        PGenTree pRoot = (PGenTree)malloc(sizeof(GenTree));
        if (pRoot == NULL)
                return NULL;
        pRoot->fatherChildIdx = -1;
        pRoot->pFather = NULL; // Means that there is no father
(1)     …

        return pRoot;
}

Result SetChildTreeAt(PGenTree pRoot, int childIdx, PGenTree pChild)
{
        Result res;
(2)     if ( … )
                return FAIL;
        // Set pChild as Child of pRoot
        res = …
(3)     if ( …  )
        {
                // Set pRoot as father of pChild
(4)             …
        }
        return res;
}
```

```
void DeleteGenTree(PGenTree pRoot)
 {
        PGenTree pFather;
        // check if tree exists
        if (pRoot == NULL)
                return;
        if (pRoot->pFather != NULL)
        {
                // Detach tree from father node (also deletes)…
(1)             …
                …
                SetAdptArrayAt(pFather->pChildrenArr, pRoot->fatherChildIdx, NULL);
        }
        else
        {
(2)             …
        }
}

// Implement Internal Functions

  …
```

שאלה 2– C++ (33 נק')

בתרגיל זה נבנה סימולטור של מערכת הנתונה על ידי גרף. גרף הוא אוסף של צמתים וקשתות המחברות בין הצמתים, כאשר כל קשת מחברת שני צמתים בלבד. נניח בתרגיל שלקשתות יש כיוון, כלומר המידע זורם בכיוון אחד בלבד, וכן משקל. בנוסף, נניח שכל צומת מפעיל פונקציה מסוימת על המידע הזורם לתוכו. הקלט לצומת יכול להגיע ממספר צמתים, והפלט יכול להיות מועבר למספר צמתים. מבנים כאלו יכולים למדל בעיות רבות, החל מקווי יצור במפעלים ועד רשתות נוירונים.

בתרגיל זה, נשתמש בגרף הבא. החצים מתארים את כיווני הקשתות והמספרים את משקלן.



לכל צומת בגרף קיימת הפעולה ()Simulate המבצעת צעד סימולציה בודד של אותו הצומת. ניתן להניח בשאלה כי סדר הסימולציה תקין, כלומר, תחילה מבצעים סימולציה על צמתי הקלט, לאחר מכן על צמתי ReLU בשכבה האמצעית ולבסוף על צומת ה-ReLU במוצא הגרף.

צומת מסוג InputVertex הוא צומת השומר תור של מספרים. מספרים מוזנים על ידי המשתמש באופן סדרתי. בכל צעד בסימולציה, הצומת מוציא כפלט את המספר בראש התור ומשחרר אותו מהתור. המספר מוכפל במשקל הקשת ומועבר כקלט לצומת בצידה השני של הקשת. את הקלט מזינים בעזרת הפונקציה PushInput באופן הבא:

```
InputVertex vInput, vInput2;
vInput.PushInput(3.1);
vInput.PushInput(4.6); //Push another value to the queue
```

צומת מסוג ReLUVertex הוא צומת אשר בכל צעד בסימולציה מקבל קלט מצמתים אחרים, ומפיק כפלט את תוצאת הפונקציה $f\left(\{x\},\{w\},bias\right)=\max\left(0,\sum_i\left(x_i\cdot w_i\right)+bias\right)$, כאשר x_i הקלטים ו- w_i משקלי הצמתים.

$bias$ הוא ערך פנימי של הצומת, ויכול להשתנות בין צמתי ReLU שונים.

ניתן לחבר צומת אחד לאחר על ידי קריאה לפעולה ConnectToVertex של הצומת, תוך ציון הצומת אליו מתחברים, ומשקל הקשת, למשל על ידי הפקודה:

```
Vertex* vLeft, vRight;

vRight.ConnectToVertex(vLeft, 3.2); // 3.2 is the weight of the connection
```

נתונה פונקצית ה-main הבאה העושה שימוש בגרף לעיל:

```
void main()
{
	Vertex* pVertex[7];

	// ----------- Setting up the graph -------------
	// Creating vertices
	// Connecting vertices

	// ----------- Pushing values to input neurons -------------

	// ------------- Simulating the net -------------
	try
	{
		for (int i = 0; i < 7; i++)
			pVertex[i]->Simulate();
	}
	catch (char* str)
	{
		cout << "Exception caught! Exiting program ..." << endl;
		exit(1);
	}

	// ------------- Printing simulation summary -------------
	cout << "The result of the simulation is: " << pVertex[6]->GetOutput() << endl;
	cout << "The net consisted of: " << endl;
	for (int i = 0; i < 7; i++)
		cout << pVertex[i] << endl;
}
```

הפונקציה מדפיסה את הפלט הבא:

```
The result of the simulation is: 15.676
The net consisted of:
ID: 1 , Type: InputVertex , Tag: Input 1
ID: 2 , Type: InputVertex , Tag: Input 2
ID: 3 , Type: InputVertex , Tag: Input 3
ID: 4 , Type: InputVertex , Tag: Input 4
ID: 5 , Type: ReLUVertex , Tag: HL1-1 , Bias: 3
ID: 6 , Type: ReLUVertex , Tag: HL1-2 , Bias: 0
ID: 7 , Type: ReLUVertex , Tag: Output 1, Bias: 2
```

א.

א. (13 נק')
הגדירו את המחלקות בבעיה. ההגדרה חייבת לכלול את כל המתודות והמשתנים החברים הנדרשים על מנת שקוד ה-main ירוץ, וכן כל פונקציה נוספת. אין צורך לממש את המתודות.
לנוחיותכם, הגדרה **חלקית** של המחלקה Vertex:

```
class Vertex
{
protected:
      std::vector<Vertex*>
m_connectedVertices;
      std::vector<double> m_weights;

      int m_id;
      double m_output;
      char* m_tag;
};
```

ב. (13 נק')
ממשו את הפונקציה ()Simulate בכל המחלקות הרלוונטיות. אין למשש פונקציות מיותרות.

ג. (7 נק')
כעת אנו מעוניינים להכליל את הגרף כך שיוכל לטפל במידע מורכב יותר. הוצע להוסיף תבניות למחלקות שהגדרתם (template <class T>) במקום הגדרת ה-double. כלומר הקלטים, הפלטים וה-bias יהיו כעת מסוג T כללי. המשקל של הקשתות נשאר double כמקודם. אילו פונקציות חייב לממש אובייקט המחליף את T? ספקו הצהרה בלבד. אין להצהיר על פונקציות מיותרות.

להלן רשימה של פונקציות שימושיות עבור המיכל std::vector:

1) void vector::clear() – ריקון המערך.
2) void vector::push_back(T val) – הוספת ערך לסוף המערך.
3) void vector::pop_back() – הוצאת ערך מסוף המערך.
4) int vector::size() – אחזור גודל המערך.
5) bool vector::empty() – בדיקה האם המערך ריק.
6) T& vector::operator[](inti dx) – גישה לתא בעל האינדקס idx.

להלן רשימה של פונקציות שימושיות עבור המיכל std::queue:

7) void queue::push(T val) – הוספת ערך חדש לתור.
8) void queue::pop() – מחיקת הערך הישן ביותר מהתור.
9) T& queue:front() – אחזור הערך הישן ביותר בתור.
10) int queue::size() – אחזור גודל התור.
11) bool queue::empty() – בדיקה האם התור ריק.

שאלה 3 – C++ (15 נק')

1. (8 נק')

נתון קובץ המנשק של תבנית המחלקה Stack מתרגול 12:

```
template <class T, int size>
class Stack {
public:
        Stack() : Num_(0) {}
        bool Push(const T& item);
        T Pop();
        bool IsFull() { return (Num_ == size); }
        bool IsEmpty() { return !Num_; }
private:
        T Arr_[size];
        int Num_;
};

template <class T, int size>
bool Stack<T, size>::Push(const T& item){
        if (IsFull())
                return false;
        Arr_[Num_++] = item;
        return true;
}

template <class T, int size>
T Stack<T, size>::Pop() {
        if (IsEmpty())
                throw "Can't pop – Stack is empty!";
        return Arr_[--Num_];
}
```

נתונה מחלקה שמממשת מכוניות:

```
class Car {
public:
        Car(int LicensePlate, const char* Manufacturer);
private:
        int LicensePlate_ ;
        char* Manufacturer_;
};
```

א. השלם את הצהרת המחלקה Car כך שהקוד הבא יעבוד כשורה:

ב. כתוב מימוש <u>לאחת</u> מהמתודות שהוספת למחלקה Car.

```
int main(){
        Stack<Car, 5> Car_Stack;
        Car binba(12345678,"Honda");
        Car Bigfoot(12344321,"BMW");
        Car_Stack.Push(binba);
        Car_Stack.Push(Bigfoot);
        Car ototo = Car_Stack.Pop();
        Car McQueen(54378966,"Volvo");
        Car_Stack.Push(McQueen);
}
```

הערה: אין צורך לרשום מחדש את הצהרות המחלקות, אלא רק את מה שמשתנה.

2. (7 נק')

נתונה לכם גרסא רזה של המחלקה Date מתרגול 7:

```
class Date{
public:
        Date(int day, int month, int year);
        int theDay() const;
        int theMonth() const;
        int theYear() const;
private:
        int day_;
        int month_;
        int year_;
};
```

בנוסף נתונות 3 מחלקות לשמירת אירועים ביומן העושות שימוש ב-Date: Event, Meeting and Vacation

```
class Event {
public:
        Event(const char * Title, const Date& StartDate);
private:
        const char * Title_;
        Date StartDate_;
};

class Meeting : public Event {
public:
        Meeting(const char * Title, const Date& StartDate, const char * Location, const char* WithWho);
private:
        char* Location_;
        char* WithWho_;
};
class Vacation : public Event {
public:
        Vacation(const char * Title, const Date& StartDate, const char * Location,int numDays);
private:
        char* Location_;
```

```
        int NumDays_;
};
```

א. השלם את הצהרות המחלקות (Event,Meeting and Vacation) והוסף פונקציות כדי שהקוד הבא יעבוד כשורה.

ב. כתוב מימוש <u>לכל</u> הפונקציות והמתודות שהוספת:

```
int main() {
  Event* EventArr[5];
  EventArr[0] = new Meeting("Lunch", Date(22,3,2016), "Cofix", "Yossi");
  EventArr[1] = new Event("MAMAT exam", Date(16,2,2016));
  EventArr[2] = new Meeting("Project Status", Date(21,6,2016), "Technion", "Haim");
  EventArr[3] = new Vacation("Trip with the family", Date(22,6,2016), "Greece", 4);
  EventArr[4] = new Vacation("Diving", Date(11,8,2016), "Eilat", 4);
  for(int i=0;i<5;i++) {
        cout<< *EventArr[i] <<endl;
  }
}
```

<u>הערות:</u>

א. אם יש מספר מימושים שונים אפשריים אז בחר אחד

ב. אין צורך לרשום מחדש את הצהרות המחלקות, אלא רק את מה שהוספת.

ג. אל תשנה את ה-main!

שאלה 4 - BASH (14 נק')

נתון קובץ בשם salaries המפרט את המשכורות החודשיות של עובדי חברה מסוימת. השורה הראשונה נראית כך:

num job id salary

כאשר מילים בשורה מופרדות ע"י הרווחים.

ושאר השורות מפרטות את פרטי המשכורות. שורות לדוגמא:

1 engineer 034524566 8000
2 manager 122334455 15000
3 engineer 156254132 9000
4 student 142635472 5000
…

כאשר גם כאן מילים בשורה מופרדות ע"י הרווחים. שדה ראשון הינו מספר סידורי של העובד, שדה שני – תפקיד, שדה שלישי – מספר ת.ז., ושדה רביעי – גובה השכר.

א. (6 נק')
רשום סקריפט ב-BASH בשם average. הסקריפט יקבל שם הקובץ עם משכורות כפרמטר וידפיס את המשכורת הממוצעת של כל העובדים. למשל הרצת הסקריפט average עם קובץ salaries שבדוגמא:

average salaries

תדפיס: 9250

ב. (8 נק')
רשום סקריפט ב-BASH בשם average_per_job. הסקריפט יקבל כפרמטר שם הקובץ עם המשכורות ורשימת תפקידים, וידפיס את השכר הממוצע של כל אחד מהתפקידים. למשל עבור ההרצה:

average_per_job salaries engineer manager

ההדפסה תהיה:

average salary of engineer: 8500
average salary of manager: 15000

<u>הערות</u>:

- אין להשתמש בקבצי ביניים.
- **בסעיף א'** חלקו את הפתרון לשני סקריפטים קצרים, הראשון בשם average ובאורך של שורה אחת, והשני בשם calc_average ובאורך של עד 10 שורות.
- **בסעיף ב'** חלקו את הפתרון לשני סקריפטים קצרים, הראשון בשם average_per_job באורך של עד 7 שורות ובתור סקריפט שני תשתמשו ב- calc_average **שכתבתם בסעיף א'**.
- אין צורך לרשום את השורה #! /bin/bash
- על מנת לחשב את השכר הממוצע תשתמשו בחלוקת מספרים שלמים (integer division).
- שים לב שעליך להשתמש בסקריפט calc_average גם בסעיף ב', לכן תתכנן אותו בהתאם.