מבוא לתכנות מערכות 10010 סמסטר א' – תש"פ

תרגיל בית מס' 3

נושא התרגיל: קבצים, רשימות מקושרות, מצביעים לפונקציות, Variadic Functions

יש להגיש אך ורק דרך תפריט המטלות שבאתר הקורס, כפי שהוסבר בתרגול.

**ניתן לעבוד בזוגות**

הנחיות הגשה כלליות:

* התרגיל ייבדק בסביבת MS Visual Studio
* הקוד חייב לעבור קומפילציה, קוד שאינו מתקמפל לא ייבדק
* יש להגיש את כל תיקיית הפרויקט, מקובצת לקובץ zip/rar/tar ששמו כשם הסטודנט. שם פרטי ומשפחה.

**הוראות כלליות:**

1. יש להקפיד על כללי הנדסת התוכנה:
   1. מבנה התכנית (הזחות) ותיעוד במידת הצורך.
   2. חובה להשתמש בקבועים במקומות המתאימים.
   3. יש להשתמש בפונקציות קצרות, כלליות, קריאות ושימושיות.
   4. יש להקפיד על בדיקת תקינות קלט.
   5. הפלט צריך להיות כפי שניתן בתרגיל.
   6. קוד קצר, לא מסורבל ויעיל הן מבחינת כתיבתו והן מבחינת ריצת התוכנית.
2. יש לחלק את הפרויקט לקבצים. חשוב איך לארגן את הקוד בקבצים השונים.
3. יש לקרוא את ההוראות במדויק.

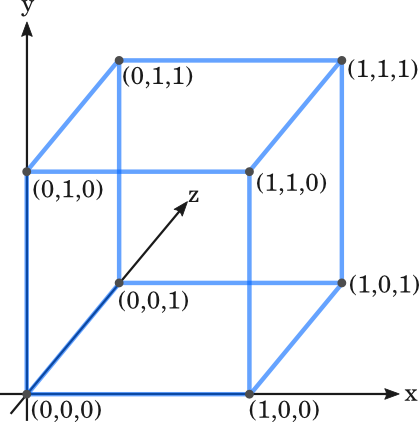
**פירוט התרגיל**

בתרגיל זה יש לבצע מספר פעולות על קבצים המתארים צורות של גופים תלת ממדיים במרחב. משתמשים בקבצים האלו בתחום הדפסה תלת ממדית, תצוגה של מציאות מדומה ועוד.

מספר חברות פיתחו יישומים ליצירת גופים תלת ממדיים במרחב ווירטואלי. רובם מגדירים את הגוף כאוסף מצולעים ה"עוטפים" אותו. לצורך כך מגדירים מספר רב של נקודות (**vertexes**)על הגוף ובנוסף מגדירים את המצולעים (**faces**)שהקודקודיים שלהם הם מהאוסף של הנקודות שהוגדרו קודם.

לדוגמה, את הקובייה עם גודל הצלע 1 אפשר להגדיר כך:

* הקודקודיים (vertexes) של הקובייה, בכול שורה מופיעות קואורדינטות של הנקודה :



**8**

**6**

(0, 0, 0)

(0, 1, 0)

**4**

(1, 0, 0)

**2**

(1, 1, 0)

**7**

(0, 0, 1)

**5**

(0, 1, 1)

(1, 0, 1)

(1, 1, 1)

**1**

**3**

* המצולעים (faces) של הקובייה, בכול שורה מופיעות מספרים סידוריים של הנקודות שבקודקוד של המצולע:

1 2 4 3

3 7 8 4

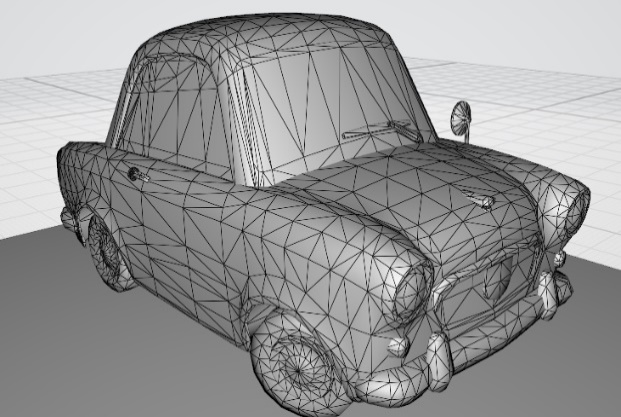
1 5 6 2

7 5 6 8

4 8 6 2

3 7 5 1

ברוב המקרים בתור המצולעים ישתמשו במשולשים וכך משיגים את הדיוק והפירוט הנדרשים





קיים מגוון רחב של פורמטים שונים של קבצים לתיאור גופים - STL, OBJ, FBX, DAE

VRML ואחרים. אפשר להוריד מהרשת מודלים של אובייקטים תלת ממדיים:

<https://free3d.com/3d-models/obj>

<http://www.cadnav.com/3d-models/sort-17.html>

ובעזרת תוכנות המציגות ( (viewersאפשר להתרשם מהוויזואליזציה של האובייקטים ואף לערוך אותם.

<https://3dviewer.net/>

<https://www.open3mod.com/>

<http://www.tucows.com/thankyou.html?swid=605254>

**בתרגיל זה נתייחס לאובייקטים המוגדרים בפורמט OBJ.**

<https://en.wikipedia.org/wiki/Wavefront_.obj_file>

הקובץ בפורמט זה הוא קובץ טקסט המכיל מידע הנדרש לוויזואליזציה של אובייקט, בין היתר מידע על vertexes ו- faces.

* כל שורה בקובץ שמתחילה באות **v** עם שלושה מספרים מתארת נקודה בחלל (x, y, z). לדוגמה, שורה

**v 5.3674 0.607756 6.01734**

מתארת נקודה במקום **(5.3674, 0.60775, 6 6.01734).** לכול נקודה, vertex יש מספר סידורי שהוא שווה לכמות הנקודות שהוגדרו בקובץ לפני הנקודה P פלוס 1. משתמשים במספר הזה בהגדרת face.

* כל שורה שמתחילה באות **f** ואחריה מופיעים מספרים שלמים מתארת מצולע (face) מסוים באמצעות נתוני הקודקודים שלו. חלק מהנתונים יכולים לא להופיע, אך תמיד כן מופיע מספר סידורי של vertex (מופיע מודגש בדוגמאות הבאות למטה). לדוגמה, משולש עם קודקודים בנקודות שהמספרים הסידוריים שלהן הם**1842, 1841**  **1835,** יכול להופיע באחת מהצורות:

**f**  **1835/**1966**/**1855 **1842/**1976**/**1865 **1841/**1974**/**1863

**f**  **1835/**1966 **1842/**1976 **1841/**1974

**f**  **1835//**1855 **1842//**1865 **1841//**1863

**f**  **1835** **1842** **1841**

**הערה**: שים לב לכמות ולמיקום של הקווים הנטויים בשורות הראשונה, השנייה והשלישית. כמות הרווחים בין המספרים לא מוגדרת מראש, אך יש רווחים. שני קווים נטויים אחד אחרי השני הם צמודים, ללא רווח. בנוסף יתכן ו face מסוים לא יהיה משולש כלומר יורכב מיותר משלוש vertexes.

**הגדרה**: אוסף אובייקטים נקרא Scene. לצורך התרגיל נדרש לממש את האוסף בעזרת רשימה מקושרת חד כיוונית.

**לצורך פתרון התרגיל נגדיר את המבנים הבאים:**

typedef struct {

float x; // Coordinates of Vertex

float y;

float z;

} Vertex;

typedef struct {

int size; // Number of vertexes of the face

int \*vertex; // Array of Ids of the vertexes

} Face;

typedef struct {

// Array of all Vertexes

int numberOfVertexes;

Vertex \*vertexes;

// Array of all Faces

int numberOfFaces;

Face \*faces;

} Object;

typedef struct {

// This structure contains a Single Direction Linked List of all objects in the Scene

} Scene;

enum FileType { TextFormat, BinaryFormat };

**משימות התרגיל**

1. כתוב פונקציה

**void transformObject(char \*originalObjectFileName, char \*deformedObjectFileName);**

הפונקציה מבצעת טרנספורמציה על קובץ Obj נתון. היא מקבלת שמות של שני קבצים בפורמט OBJ. המקורי וזה שיכיל את הטובייקט לאחר טרנספורמציה על כל Vertex שהוגדר בקובץ המקורי לפי החוקיות הבאה:

xnew = 0.3\*x

ynew = y

znew = z

1. כתוב פונקציה שמקבלת שם של קובץ בפורמט OBJ ובונה מבנה Object בהתאם.

**Object \* createObject (char \*filename)**

1. כתוב פונקציה שבונה מבנה Scene המורכב ממספר אובייקטים כרשימה מקושרת.

**Scene \*createScene(char \*fileName, ...);**

הפונקציה המקבלת מספר לא ידוע מראש של מצביעים לשמות קבצים, בכל קובץ הגדרה של אובייקט בפורמט OBJ. הפרמטר האחרון הוא NULL. הפונקציה יוצרת ומחזירה מבנה מסוג Scene שמכיל רשימה מקושרת חד כיוונית של כל האובייקטים.

1. כתוב פונקציות שמבצעות פעולות שונות על האובייקט. החתימה האחידה של הפונקציות תעזור בכתיבה גנרית בסעיף הבא.
   1. פונקציה המקבלת את המצביע על ה-Object ומחשבת את מספר ה-faces של האובייקט להם שלושה Vertexes .

**void printFaces(Object \*ptr, void \*numberOfTriangularFaces)**

* 1. פונקציה המקבלת את המצביע על ה-Object מחזירה את מספר ה-vertexes של האובייקט

**void printVertexes(Object \*ptr, void \*numberOfVertexes)**

* 1. פונקציה המקבלת את המצביע על ה-Object מחשבת את השטח כולל של כל ה-faces של האובייקט להם שלושה Vertexes.

**void getTotalArea (Object \*ptr, void \*totalAreaOfTriangularFaces)**

1. כתוב פונקציה **perform** המקבלת ,Scene מצביע לפונקציה מהסוג שהוגדרו בסעיף 4 , מחרוזת המציינת את סוג משתנה ההדפסה, ומצביע למחרוזת . הפונקציה perform עוברת על כל האובייקטים ב-Scene ומבצעת קריאה בעזרת המצביע לפונקציה. בסוף היא מדפיסה את המחרוזת ואת הסכום של כול התוצאות. סוג ההדפסה האפשרי INT, DOUBLE
2. כתוב פונקציות ששומרת/מעלה את ה-Scene בקובץ/מקובץ טקסט או בינארי לפי בחירת המשתמש. אין דרישה ספציפית לפורמט הקובץ כל עוד הקריאה של הקובץ שנשמר מייצרת חזרה Scene תקין

**void saveScene(Scene \*scene, char \*fileName, enum FileType type);**

**Scene \* loadScene(char \*fileName, enum FileType type);**

בפונקצייה loadScene נעשו כל הקצאות הזיכרון הנדרשות ביצירת כל המבנים החדשים המרכיבים את ה-scene החדש שנוצר.

1. כתוב פונקציה שמשחררת את כל הזיכרון שהוקצה במהלך בניית ה-scene

**void freeScene(Scene \*scene)**

1. בדוק את הפונקציות שכתבת בעזרת התוכנית הבאה הוסף בדיקות תקינות כך שהתוכנית לא תעוף במידה לדוגמא שהקובץ obj לא נמצא:

void main() {

Scene \*scene;

Scene \*scene1, \*scene2;

// Create a Scene of Objects received

// from files GoingMerry.obj, Donald.obj, Pony\_cartoon.obj

scene = createScene("GoingMerry.obj", " Donald.obj", " Pony\_cartoon.obj", NULL);

// Print out all Vertexes

perform(scene, printVertexes, "INT", "The number of vertexes of all objects is");

// Count the number of all Triangular faces of all objects in Scene

perform(scene, printFaces, "INT", "The number of Triangular faces of all objects is");

// Count the total area of all Triangular faces of all objects in Scene

perform(scene, getTotalArea, "DOUBLE", "The total area of Triangular faces of all objects is");

// Save the world to file in Text and in Binary formats

saveScene(scene, "FirstScene.dat", TextFormat);

saveScene(scene, "SecondScene.dat", BinaryFormat);

// Retrieve the saved data

scene1 = loadScene("FirstScene.dat", TextFormat);

scene2 = loadScene("SecondScene.dat", BinaryFormat);

// Check Retrieve the saved data of text format

perform(scene1, printVertexes,”INT” "The number of vertexes of all objects is");

perform(scene1, printFaces,”INT”, "The number of Triangular faces of all objects is");

perform(scene1, getTotalArea,”DOUBLE”, "The total area of Triangular faces of all objects is");

// Check Retrieve the saved data of binary format

perform(scene2, printVertexes,”INT” "The number of vertexes of all objects is");

perform(scene2, printFaces,”INT”, "The number of Triangular faces of all objects is");

perform(scene2, getTotalArea,”DOUBLE”, "The total area of Triangular faces of all objects is");

// Make transformation based on the original object defined in file Donald.obj

transformObject("Donald.obj", “StrangeDonald.obj”);

// Free all memory allocated for the Scene

freeScene(scene);

freeScene(scene1);

freeScene(scene2);

}