**מדעי המחשב**

**פרוייקט גמר**

**אפליקציית מחשבון לוגי**

**מבצעים:**

**בשארה חמאתי ~ 204524185**

**אסלאם אכתילאת ~ 206143166**

**העבודה בוצעה בהנחיית:**

**ד''ר דאני קוטלר**

**2019/2020**

**תוכן עניינים**

|  |  |
| --- | --- |
| 3 | הקדמה |
| 4 | מטרת העבודה, הצגת הבעיה וניתוח מערכת |
| 5 | זרימת האינפורמציה |
| 6-7 | מבני וארגון הקבצים |
| 8 | שלבים בפיתוח הפרויקט |
| 9 | מגבלות וקשיים |
| 10 | מה שפרנו וחדשנו !! |
| 11-12 | אלגוריתמיים |
| 13 | בדיקות |
| 14 | סיכום |
| 15 | תמונות |
| 16 | ביבליוגרפיה |

**הקדמה**

כסטודנטים שנה ג , רצינו פרויקט יעיל ועוזר לסטודנטים אחרים וגם למרצים, מתוך הרשימה החלטנו לבחור ולעשות מחשבון לוגי אבל בצרוה מחודשת.

יש כבר מחשבון לוגי באתרי אינטרנט וגם אפליקציות ל ווינדוס , לכן בחרנו לעשות מחשבון לוגי לאנדרואיד.

המחשבון הוא לוגי מתמטי שמאפשר לחשוב פעולות בסיסיות ביעילות ובשיטה רקורסיבית.

המחשבון יכול לחשב ולהציג את טבלת האמת של ביטויי פסוקים לוגים, הוא גם מחשב צורות נורמליות כפי שלמדנו בקורס (CNF,DNF) בצורה הכי פשוטה שיש .

בנוסף, אפשר באמצעותו לבדוק אם קיימת שקילות או גרירה סמנטית בין פסוקים שונים.

הרעיון הוא לעשות אפליקציה סלולרית לא תלויה באינטרנט (לא צריך אינטרנט לשימוש) וקלה לשימוש לעזור לסטודנטים לבדוק את תשובתם בזריזות וגם למרצים.

**מטרת העבודה, הצגת הבעיה וניתוח מערכת**

המטרה היא לעשות פרויקט שימושי ויעיל כך שיהיה אפשרי לכל בן אדם להשתמש ולהיעזר בו כגון סטודנטים , מרצים ובתי-ספר...

בקורס הלוגיקה , לעשות טבלת אמת או עץ בינארי או למצוא צורה נורמלית לביטוי ארוך ( עם מספיק אטומים) נחשבת כעבודה שחורה , ארוכה וקל להתבלבל בתשובה. לכן הפרויקט הזה היה מתאים למטרה שלנו.

במהלך העבודה ובהנחיית ה-מנטור שלנו ,יצאנו ברעיון לעזור גם למרצים , הוספנו אופציית העתקת טבלת האמת בכדי שיעזור להם בכתיבת תרגילי בית או פתרונות למבחנים.

עוד משהו שהסגנו תוך כדי העבודה, היא להתחזק בשפת JAVA וגם לעשות אפליקציות סלולריות יעילה מבחינה זמן ריצה וגם מבחינת שימוש (קלה ו שימושית) .

**זרימת האינפורמציה**

המשתמש פותח את האפליקציה , ואז לוחץ כל כפתור "Calculator"

1. המשתמש יועבר למסך בו הוא יכול לבחור האופציה שהוא רוצה קודם ( טבלת אמת, CNF,DNF,..)
2. אחר כך הוא יכול להדפיס את הפסוק ע''י כפתורים קיימים בחלון של אותיות לאתניות ואופרטורים, ואז הוא לוחץ על כפתור PROCCESS .
3. התוכנה בודקת האם הפסוק הוא פסוק רגיל שאין בו סימני סמנטיקה כמו (⊣⊢, ⊨).
4. בשלב זה בודקים האם הקלט נכון, אם כן מייצרים עץ בינארי של הפסוק אחרת מחזירים הודעת שגיאה מתאימה למשתמש.
5. באם הפסוק רגיל, בונים את טבלת האמת של העץ שקבלנו מסעיף הקודם, גם מחשבים את הצורות הנורמליות שהם CNF, DNF אחר כך מפשיטים את הפסוקים שקיבלנו באם אפשר לצורה ברורה וקצרה יותר.
6. אם יש בפסוק סימן של שקילות (⊣⊢) אז פועלת פונקציה שבודקת האם יש שקילות בין שני הפסוקים, אם כן מציגים למשתמש טבלת אמת ותשובה סופית אם יש שקילות או לא.
7. אם בפסוק יש סימן של גרירה (⊨) אז יתכן שיש יותר מנתון אחד במשוואה.

מסדרים את הנתונים ובודקים האם יש גרירה סמנטית בין הצד השמאלי והצד הימני, מציגים טבלאות אמת לצד ימין (ההנחה) ו צד שמאל (המסקנה) , ובסוף יש תשובה סופית אם קיימת גרירה סמנטית או לא.

**מבני וארגון הקבצים**

* קבצי מקור ב JAVA
  + About.java
  + AtomList.java
  + BinaryTreePrinter.java
  + Constants.java
  + Expression.java
  + FormalConverter.java
  + HomePage.java
  + Howto.java
  + MainActivity.java
  + Processor.java
  + Result.java
  + RPNConverter.java
  + RPNEvaluator.java
  + TokenTreeNode.java
  + TruthTable.java
* קבצי עיצוב ב XML
  + activity\_about.xml
  + activity\_home\_page.xml
  + activity\_howto.xml
  + activity\_main.xml
  + activity\_result.xml
* קבצי ערכים ב XML
  + colors.xml
  + strings.xml
  + styles.xml
* קבצי גלריה (תמונות)
  + H1.jpg
  + H2.jpg
  + H3.jpg
  + H4.jpg
  + H5.jpg
  + H11.jpg
  + Icon.jpeg
  + Logohd.jpg

**שלבים בפיתוח הפרויקט**

* המחשבה הראשונה שעלתה לראש הוא איך נתחיל ?? איך מממשים את הפונקציות ?? איזה אלגוריתמים צריכים ?? האם שפת JAVA מספיקה לכל הדרישות ונוחה לממש את הפרויקט שלנו ??
* הייתה דרישה לממש מחשבון לוגי ובחרנו לעשות את המחשבון כאפליקציה סלולרית.

ו השתמשנו בשפת JAVA משום שהיא שפה חזקה, יעילה ו מאפשרת להשתמש ברקורסיה .

* אחר כך הצטרכנו לבחור סביבת עבודה ובחרנו אפליקציית Android-studio, משום שהשתמשנו בה הקורס האנדרואיד והיא סביבה די חזקה לאנדרואיד.
* קודם כל עיצבנו את האפליקציה, והכנו את כל הכפתורים שהיינו צריכים למימוש נוח .
* אחרי כך הקוד של קלט פלט, השקענו הרבה בזה כי מאמנות ומניסיון עצמי ידענו שזה החלק היותר חשוב שיעזור לנו בקוד ובאלגוריתמים יותר קלים ,קצרים ורקורסיביים .
* השלב הבא היה לקרוא ולחפש השיטה הכי יעילה לדרישות ולממש את הפונקציות בצורה ברורה. הוא השלב הכי קשה לקח המון זמן עד שהצלחנו לכתוב כל הפונקציות.
* השלב האחרון , הוא שלב הבדיקות שהיה צריך הרבה בדיקות וחיפושים באינטרנט כדי לוודא שאין טעיות חישוביות .

**מגבלות וקשיים**

* בניית פונקציית פישוט לצורות נורמלית בצורה רקורסיבית ויעילה.
* בדיקת קלט ולהחליט אם הקלט שגוי.
* לימדה עצמית על כל מני דרכים חדשים בסביבת מערכת סלולרית.

הדבר הזה חשוב מאוד , כאשר אתה לומד שפה חדשה אתה חייב ללמוד את הקיצורים ,הפונקציות המוכנים ,ואת הדברים הבסיסים.

* בחירת עיצוב נוחי למשתמש .
* בניית העץ הבינארי של צורות נורמליות.

**מה שפרנו וחדשנו !!**

* טבלת אמת של הפסוקים :

טבלת אמת ברורה עם אפשרות להעתקה, אין הגבלה בכמות האטומים ובגודל הטבלה .

* מחשבון קל לשימוש ולכתיבת פסוקים ( הכל לפי כפתורים לא צריך לחפש סמלים)

זה הרעיון שמחפש עליו כל סטודנט בקורס הלוגיקה .

* שקילות וגרירה סמנטית

האפליקציה שלנו תומכת גם במציאת השקילות והגרירה.

* צורות נורמליות פשוטות (CNF,DNF) כולל עץ בינארי.
* שימוש בלי אינטרנט:

המשתמש לא חייב להיות מחובר לאינטרנט ו הוא עדיין יכול לחשב ולקבל התוצאות.

**אלגוריתמיים**

convertToCNF:

לאלגוריתם שלושה שלבים:

1. הפרוצדורה IMP\_FREE: החלפת כל הופעה של → בביטוי שקול בעזרת הזהות 
2. הפרוצדורה NNF (negation normal form): הבאה למצב שהקשר ¬ נמצא רק בליטרלים באמצעות זהויות דה-מורגן והזהות 
3. הפרוצדורה CNF פועלת רקורסיבית באופן הבא:
   1. אם φ בצורת CNF אז CNF(φ)=φ
   2. CNF(φ1∧φ2)=CNF(φ1) ∧CNF(φ2)
   3. CNF(φ1∨φ2)=DISTR(CNF(φ1) ,CNF(φ2))

distribute :

הפרוצדורה DISTR פועלת רקורסיבית באופן הבא:

נניח שרוצים לחשב את DISTR(η1,η2).

1. DISTR(η11∧η12,η2)=DISTR(η11,η2)∧DISTR(η12,η2)
2. DISTR(η1, η21∧η22)=DISTR(η1,η21)∧DISTR(η1,η22)
3. אחרת (אין ∧) - DISTR(η1,η2) = η1∨η2

convertToDNF:

לאלגוריתם שלושה שלבים:

הראשון והשני הם אותם שב CNF

3. הפרוצדורה DNF פועלת רקורסיבית באופן הבא:

1. אם φ בצורת DNF אז DNF(φ)=φ
2. DNF(φ1∨φ2)=DNF(φ1)∨DNF(φ2)
3. DNF(φ1∧φ2)=DISTR(DNF(φ1) ,DNF(φ2))

distribute :

הפרוצדורה DISTR פועלת רקורסיבית באופן הבא:

נניח שרוצים לחשב את DISTR(η1,η2).

1. DISTR(η11∨η12,η2)=DISTR(η11,η2)∨DISTR(η12,η2)
2. DISTR(η21∨η22,η2)=DISTR(η1,η21)∨DISTR(η1,η22)
3. אחר (אין ∨) - DISTR(η1,η2) = η1∧η2

TruthTable:

אלגוריתם זה מייצר ו ממלא טבלת אמת בוליאנית בצורה רקורסיבית, כך שעובר על כל שורה ממלא את ערכי האטומים ב true ו- false ומחשב את ערך העמודה האחרונה שמכילה את תוצאת ההשמה של ערכי האטומים שחשבנו מקודם בפסוק הנתון.

:Proccess

אלגוריתם זה בודק את הקלט , אם אין בו סמלי גרירה או שקילות אז הוא עושה טבלת אמת רקורסיבית, אחרת הוא בודק אם המשתמש בודק גרירה סמנטית או שקילות , ואז הוא מכין טבלאות אמת בהתאם ועושה את ההשוואות.

**בדיקות**

* כל הפסוקים שלמדנו בעבר
  + - * פסוקים ממבחנים
      * פסוקים משיעורי בית
      * פסוקים מהאינטרנט
* בדיקת כלי הפישוט
  + - * בדיקה לכל כלל לבד
      * בדיקה מעורבת כל שני כללים ויותר באותו פסוק
* בדיקות קלט
  + - * בדיקות סוגריים
      * עדיפות
      * אטומים ושמות חוקים
      * קשרים חוקים
* בדיקת גרירה ושקילות סמנטית
* בדיקות של המונחה

**סיכום**

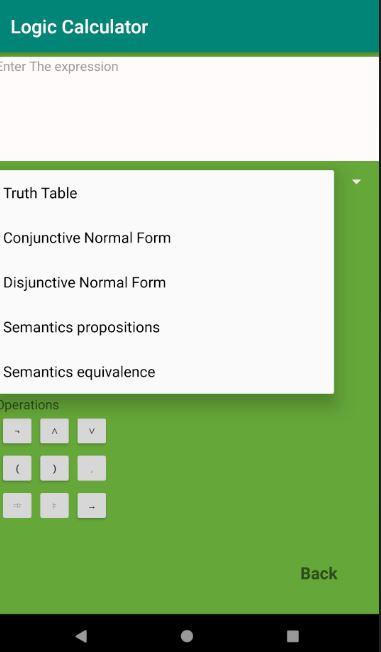
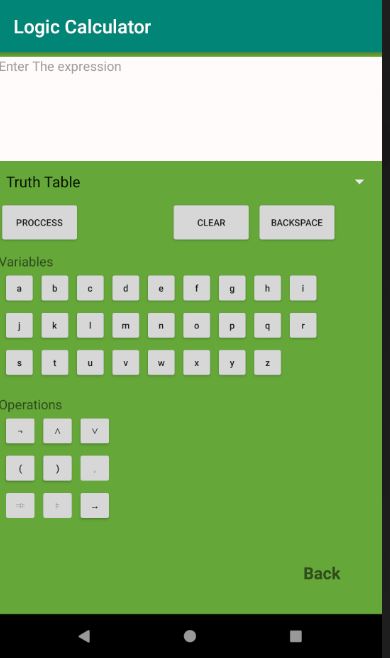
בפרויקט הזה שיפרנו את היכולת שלנו לעבוד יחד, השתמשנו ב GITHUB לצורך זה, גם

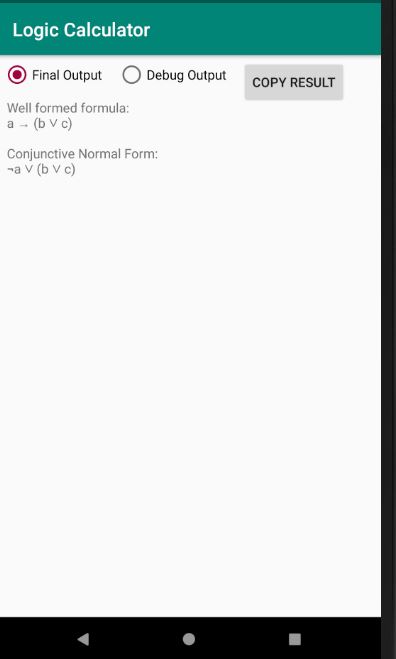
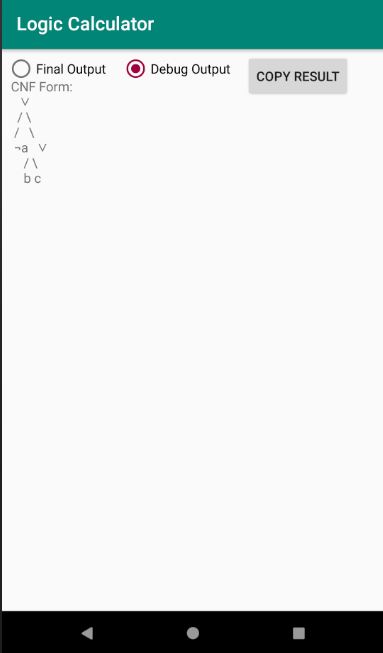
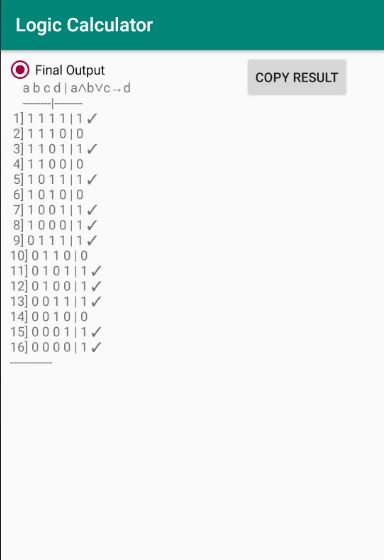
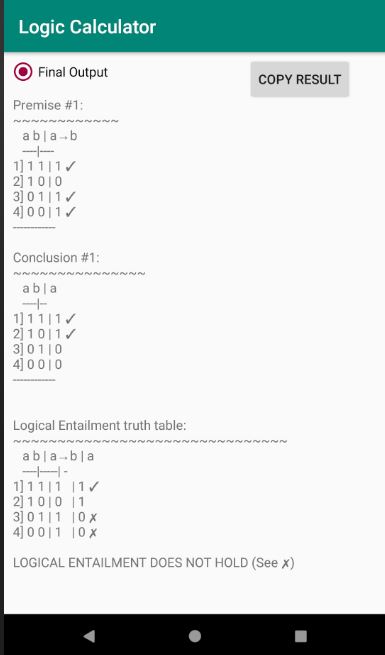
השגנו ידע נוסף לגבי אפליקציות סלולרית, סביבת עבודה חדשה (ANDROID STUDIO) , ולהתחזק בשפת JAVA ובשיטות רקורסיביות.

יצאנו בסוף באפליקציה שימושית , נוחה למימוש ו מאמנים שהסטודנטים ו אולי מרצים הולכים להשתמש בה .

היה קשה , כיף ו נהדר לעבוד על פרויקט כזה עם מטרה יחידה והיא לעזור לאנשים.

**תמונות**

****

****

ביבליוגרפיה

<https://www.cs.bham.ac.uk/research/projects/lics/>

<https://developer.android.com/>