1.2

* 4. המשתמש מכניס פקודה אחת ושם של קובץ המופרדים ביניהם עם << , ה-shell מבצע את הפקודה כך שה-output של הפקודה ילך לקובץ (במקום ל-terminal) , אם הקובץ לא קיים, אז יוצרים אותו, אם קיים the output appended. ה-shell מחכה שהפקודה תסתיים לפני שהוא ממשיך.

**הנחות:**

1. אם שורת הפקודה מכילה | אז | מופיע פעם אחת וזה ממקום נכון, כלומר יש מילה אחת לפני ואחרי ורווח לבן מפריד ביניהם
2. אם שורת הפקודה מכילה & אז זו המילה האחרונה בשורת הפקודה
3. אם שורת הפקודה מכילה << אז היא מופיע לפני המילה האחרונה בשורה (לפני שם הקובץ)
4. שורת פקודה תכיל או אחד מהסמלים: | , &, <<, או אף אחד מהם
5. שורת פקודה לא תכיל מרכאות

**פירוט המטלה:** הפונקציות למימוש

2.1 : **int prepare(void)**

השלד שמגיעה במטלה קורא לפונקציה זו לפני הקריאה הראשונה ל-process\_arglist(). הפונקציה מחזירה 0 בהצלחה, כל ערך אחר מסמן שגיאה. עליך להשתמש בזה לכל אתחול והכנה שאתה חושב שדרושים לפונקציה process\_arglist() . אם לא צריך אתחול אז שיחזיר רק 0

2.2: **int process\_arglist(int count, char \*\*arglist**)

**קלט**: מערך עם count מילים שונים מ-null, המערך מכיל את שורת הפקודה המפורקת, התא האחרון, arglist[count]==null (אז גודל המערך הוא count+1)

**התנהגות:**

1. פקודות שצויינו ב-arglist צריכות להתבצע כ-child process תוך שימוש ב-fork() וב-execvp() . אם ב-arglist יש &, |, <<, אז ה-arglist לא צריך להיות מועבר ל-execvp כמו שהוא ( יותר פרטים למטה)
2. ביצוע פקודות ברקע:

* אם המילה האחרונה היא & אז יש להריץ את ה-child process ברקע, ההורה לא יחכה ל-child process אלה ימשיך לבצע את הפקודות
* אין להעביר את & כארגומנט ל-execvp()
* יש להניח כי תהליכים ברקע לא קוראים קלט

1. Piping: