בס"ד

מטלה ראשונה בתכנות מערכות 1

שם המגיש: רעי שאול, **מספר זהות-** 325390086

מענה על השאלות הפתוחות:

שאלות כלליות על תהליך יצירת קבצים / ספריות

1. קבצי h אלו קבצים המכילים:
   א. הגדרות של מבני נתונים (struct, enum, typedef)
   ב. הצהרות של פונקציות וקבועים-ללא מימוש(#define)
   ג. הקבצים משותפים למספר מודלים

קבצי Header מאפשרים לשתף הצהרות בין קבצים שונים בפרויקט. כך ניתן להשתמש בפונקציות או במבנים שהוגדרו בקובץ אחד גם בקבצים אחרים מבלי לכתוב את ההצהרות מחדש.

קבצי c אלו קבצים אלה מכילים את **המימושים** של הפונקציות שהוכרזו ב Header-כאן כותבים את הקוד שמבצע את הפעולות.

נהוג לשים בקבצי c מימושים של פונקציות והגדרת משתנים גלובליים אם נדרש.

קבצי h.ו c.-מסייעים לשמור על עקרונות תכנות נכונים בכך שהם תורמים לארגון, לניהול ולתחזוקה של קוד, ומעודדים כתיבה מובנית וברורה. הנה כמה עקרונות מרכזיים שהם מסייעים להקפיד עליהם:

א. הפרדת ממשק ממימוש: **קובץ h.** מגדיר את ה"ממשק :(interface) "ההצהרות של הפונקציות, המשתנים והמבנים שהקוד החיצוני יכול להשתמש בהם.
קובץ c. מכיל את המימוש בפועל: כיצד הפונקציות פועלות.
ב. עקרון המודולריות- חלוקת קוד לקבצי h.ו c.-מאפשרת לפצל את התוכנית למודולים נפרדים, כאשר כל מודול אחראי לתחום מסוים
ג. צמצום כפילויות בקוד- כשכל קובץ Source כולל את אותו ,Header כל הפונקציות והמבנים מוגדרים פעם אחת בלבד בקובץ ה.Header-
ד. עקרון הסתרת מידע(Encapsulation) - ה Header-מספק גישה רק לממשק הציבורי (public)של המודול. שאר הפרטים הפנימיים (כמו משתנים פרטיים או פונקציות עזר) נשארים בקובץ ה c.-ואינם נגישים מחוץ לו.

2. שלב ראשון- יצירת קוד מקור(source Code) המכיל את המימוש בפועל- מימושים של פונקציות והגדרת משתנים גלובליים אם נדרש.
שלב שני- מקמפלים את הקוד מקור- הקומפיילר gcc מתרגם את הקוד מקור לקוד בינארי והפלט מהשלב הזה נקרא קובץ אובייקט(.o) שאינו קובץ הרצה.
שלב שלישי- קישור(linking) הלינקר משלב את קבצי האובייקט יחד עם ספריות נדרשות ויוצר קובץ הרצה. הפלט הוא מהצורה(.exe)
3. בשלב הקישור (linking)
4. יתרון של ספריה דינמית על סטטית: ספריה דינמית קטנה יותר בהרבה מגודלה של ספריה סטטית. בגלל שבדינמית יש רק העתק אחד שנשמר בזיכרון. לעומת זאת, בסטטית, תוכניות חיצוניות נבנות בקובץ הרצה.
יתרון של ספריה סטטית על דינמית: לסטטית אף פעם אין בעיות תיאום מאחר שכל הקוד נמצא במודל הפעלה אחד. לעומת זאת, בדינמית, התכוניות תלויות בספריה תואמת ואם נוריד ספריה אחת מהמערכת, התכנית לא תעבוד.
נרצה להשתמש בספריה **סטטית** כשאנחנו צריכים ליצור קובץ הרצה עצמאי (לא תלוי בספריות חיצוניות) או כשאנחנו רוצים להבטיח שגרסה מסוימת של ספריה תיכלל בתוכנית.
ספריות סטטיות גם מבטיחות ביצועים טובים יותר כיוון שהן מקומפלות ישירות לתוך קובץ

ההרצה. יתרון נוסף הוא שאין צורך לטעון את הספריה בזמן הריצה, מה שעשוי להיות חשוב במקרה של סביבות עבודה עם מגבלות זיכרון או ביצועים.

נרצה להשתמש בספריה **דינמית** כאשר אנו רוצים לחסוך בזיכרון ולהוריד את גודל קובץ ההרצה. ספריות דינמיות מאפשרות לשתף קוד בין מספר תוכניות, כך שמספר תוכניות יכולות להפעיל את אותה ספריה מבלי לכלול אותה בתוך כל קובץ הרצה בנפרד. זה גם מקל על עדכון גרסאות של ספריות – אם יש שינוי בספריה, לא נדרשת קומפילציה מחדש של כל התוכניות המשתמשות בה.

5. קבצי obj. הם קבצים בינאריים המכילים קוד אובייקט (Object Code) לאחר תהליך הקומפילציה. הפורמט של קובץ ה obj. -משתנה לפי המערכת והקומפיילר, אך באופן כללי הם כוללים:
קוד מכונה – (Machine Code) הפקודות שניתן להריץ.
טבלה של סמלים – (Symbol Table) מידע על משתנים, פונקציות, ומבנים.
מידע על קישורים – (Relocation Information) מידע על מיקומים שצריכים להתעדכן בעת הלינקינג.
מידע על דיבאג (לפי הצורך)

6. **ספריות סטטיות** :סיומת a. ב ,Unix/Linux-סיומת lib. ב.Windows
ספריות דינמיות :סיומת so. ב ,Unix/Linux-סיומת dll. ב.Windows

מענה על שאלות על הקומפיילר GCC ועל MAKEFILE

1. Wall-

2. כן, ניתן לשנות את שם קובץ ה Makefile לבחירה אישית. אם לא נרצה להשתמש בשם ברירת המחדל, נוכל לציין את שם הקובץ באמצעות הדגל f- בעת הריצה של הפקודה make. לדוגמה:

make -f re'i במקרה זה שם הקובץ makefile הוא re'i

3. **הגדרת משתנים :** מוגדרים בצורה הבאה: .NAME = value

שימוש במשתנים : השימוש נעשה באמצעות תחביר .$(NAME)

היתרון : מקל על שינוי הגדרות כלליות (למשל, שם הקומפיילר או דגלים) בכל המטרות ב.Makefile-

וגם הופך את המייק פייל לשימושי גם עבור פרויקטים אחרים.

כמו כן, קבצי מייק פייל רבים יוצרים משתנים עבור הקומפיילר, דגלים ועוד, זה יכול להיות שימוש מוגזם אבל נדע להבחין בכך.

יש משתנים מיוחדים כגון: $@, $^, $< בעלי משמעויות שונות .

4. ב makefile חוקים מופעלים כאשר make מזהה שיש צורך לעדכן קובץ יעד(target)

המבנה הבסיסי של חוק:

Target:prerequisites

Command

Target- הקובץ שצריך להיבנות

בס"ד

Prerequisite- קבצים שהמטרה תלויה בהם- כמו קבצי קוד מקור או header
command- הפקודה שתופעל כדי לייצר את ה מטרה כמו gcc

התנאים להפעלת חוק:

א. קובץ היעד חסר- אם הקובץ המוגדר כמטרה אינו קיים בתיקייה, make יבצע את החוק כדי לייצר אותו.
ב. קיים קובץ prerequisites חדש יותר מהמטרה- אם אחד מהקבצים האלה חדש יותר שונה או מעודכן לאחרונה הקובץ מטרה ייבנה מחדש.
ג. חוק מאולץ(PHONY target)- אם החוק מוגדר כחוק מדומה, האמצעות PHONY הוא יופעל תמיד ללא קשר לקיום הקובץ.
ד. תלות מרומזת(implicit rules) – אם לא מצוין חוקים מפורשים, הקובץ make ינסה להשתמש בחוקים מרומזים כמו שימוש אוטומטי ב gcc-c עבור קבצי c.

המake מחליט אם להריץ חוק בצורה הבאה:
אם program לא קיים-יופעל החוק הראשון
אם main.c שונה לאחרונה- יופעל החוק של main.o
אם utils.c או utils.h שונו- יופעל החוק של utils.o

5. makefile נחשב חכם משום שהוא: מצמצם עבודה מיותר ע"י זיהוי תלות ובודק את זמני השינוי של הקבצים.

כמו כן, הוא חוסך חזרתיות באמצעות חוקים מרומזים ותבניות. הוא גם מאפשר תחזוקה קלה ושינויים בפרויקטים בכל גודל.

6. חוקים עקיפים נוצרים כאשר makefile לא מגדיר חוקים מפורשים לבניית היעדים ואז המake משתמש בסט של חוקים מוגדרים מראש או בתבניות כלליות.(לא תמיד מספיק) החוקים העקיפים מבוססים על דפוסים ותבניות נפוצים (כמו המרת קבצי מקור .c לקבצי אובייקט .o) כך שהם מופעלים אוטומטית כאשר אין חוק מפורש.

הבעיה בחוקים אלו היא חוסר שליטה בהתנגות- חוקים עקיפים מבוססים על ברירות מחדל. אם תהליך הבניה דורש הגדרות מיוחדות כמו דגלים או שלבי עיבוד נוספים וייתכם שהחוק העקיף לא יספק פתרון הולם.

כמו כן, עם שימוש חוקים עקיפים יהיה קושי באבחון בעיות ולא נוכל למה המייק מנסה להריץ פקודה מסוימת.

מלבד זה, חוקים עקיפים יכולים להסתמך על תלות שלא הוגדרה במפורש- דבר שיכול לגרום לבעיות נוספות.

כמו כן, הביצועים נמוכים יותר כיוון שהמייק ישקיע זמן בחיפוש חוק עקיף במקום פשוט להריץ חוק מפורש.

וגם התחזוקה של חוקים אלו בעייתית, כי אם הפרויקט גדול וכולל תלויות רבות, חוקים עקיפים יכולה להקשות ממש על ההבנה של התכנית.

7. כדי לגלות את רשימת התלויות (dependencies) של קובץ קוד מקור, ניתן להשתמש ב GCC-עם האפשרות M-או דומות לה. אלה יפיקו רשימה של קבצים שהקוד תלוי בהם, כגון קבצי כותרת.(.h)

לדוג' gcc -M source.c

8. הפקודה: ar rcs libdata.a data.o stack.o list.o

בס"ד

ar- הוא כמו ארכיון של קבצי אובייקט

r: מוסיף קבצים לספרייה (מחליף קבצים קיימים אם הם כבר נמצאים).

c: יוצר את הספרייה אם היא לא קיימת.

s: יוצר אינדקס פנימי לשימוש מהיר בקישור.(linking)

העשרה

שלב האסמבליזציה- שלב האסמבלציה ממיר את קוד שפת הביניים ,(Assembly Code) שנוצר בשלב הקומפילציה, לקוד מכונה (Machine Code) שאותו מעבד המחשב יכול להבין.

Input: קוד Assembly שנוצר על ידי הקומפיילר.

Output: קובץ אובייקט o.או ,obj שמכיל קוד מכונה בינארי עבור כל פונקציה או מודול בקובץ המקור.