# קומפילציה

## קומפילציה

תהליך הקומפילציה של קוד ב-C++ מורכב משלושה שלבים:

* **קדם מעבד (Preprocessor)** - מסיר הערות ורווחים, ומחליף את כל הערכים שהגדרנו כקבועים באמצעות "define". כדי לתת פקודה לקדם מעבד נשתמש בסימון #. כל הפקודות לקדם מעבד אין להם סיומת, ולכן הפקודה נגמרת בירידת שורה.
* **מהדר (Compiler)** - מתרגם את הטקסט בשפת C שנוצר מהקדם מעבד אל שפת מכונה. הקובץ שנוצר לאחר פעולת הקומפיילר הוא object file עם סיומת .o
* **מקשר (linkage)** - מקשר את כל קבצי object שנוצרו מהקומפיילר ואת כל הספריות שהוספנו, אל קובץ אחד הניתן להרצה.

## פקודות

### בסיס

**קימפול** - כדי לקמפל את הקובץ שלנו, נניח ששמו file.cpp והקומפיילר שלנו הוא gcc, נשתמש בפקודה:

gcc file.c

התוצאה של פקודה זו תהיה קובץ בשם a.out אותו ניתן להריץ באמצעות הפקודה:

./a.out

**שינוי שם** - אם נרצה שהקובץ המוכן להרצה יקבל שם אחר, לדוגמא outputfile, נשתמש בדגל -o. הסדר פחות חשוב.

gcc file.c -o outputfile

gcc -o outputfile file.c

**בדיקת שגיאות** - כדי לקבל עוד מידע על שגיאות בקימפול אם ייווצרו, נוסיף דגל -wall.

gcc -Wall -o errorlog file.c

### הפרדת שלבים

הפקודות שהראנו לעיל מבצעות קימפול וקישור ביחד ויוצרות קובץ שניתן להרצה בפקודה אחת. נוכל להפריד את הפעולות לקומפילציה בלבד וקישור בלבד.

**קומפילציה בלבד** - פעולה זו תקמפל את הקובץ בלבד ולא תייצר קישורים עם קבצים אחרים. אין צורך לשנות את שם הקובץ שנוצר, בברירת מחדל הוא לוקח את הקובץ .c ויוצר קובץ חדש עם אותו שם .o

gcc -c file.c //output "file.o"

**קישור בלבד** - מתבצעת אך ורק על קבצי .o בסוף הפעולה יוצרת קובץ שניתן להרצה.

gcc -o outputfile file.o

### מספר קבצים

**קימפול מספר קבצים** – כדי ליצור תוכנית המורכבת מכמה קבצי מקור, נרשום את כולם אחד אחר השני ללא פסיקים.

gcc -o outputfile file1.c file2.c file3.c

אמנם בדרך כלל מקמפלים בלבד ללא חיבור כל קובץ בנפרד ולבסוף מחברים את כולם.

gcc -c file1.c

gcc -c file2.c

gcc -c file3.c

gcc -o outputfile file1.o file2.o file3.o

### הרצה

כדי להריץ קובץ יש לציין את מיקום הקובץ שרוצים להריץ, לאחריו נרשום `/` ואז שם הקובץ שרוצים להריץ. בדרך כלל נריץ מתוך התיקייה שבה נמצא הקובץ ולכן נרשום `./`, ואז שם הקובץ שרוצים להריץ.

./a.out

# Makefile

## הגדרה

דרך הקימפול והרצה של תוכנית דרך הטרמינל היא לעיתים מאוד מסובכת, במיוחד כשיש המון קבצים וכדי לקמפל צריך לכתוב כל פעם מחדש פקודות ארוכות ומסורבלות. בנוסף, בתוכנית שבה יש מספר קבצים התלויים זה בזה, כאשר משנים קובץ אחד צריך לקמפל מחדש את כל הקבצים שמתקשרים לקובץ ששינינו. ככל שהתוכנית מכילה יותר קבצים, כך בעיות אלו מקשות יותר על המתכנת.

כדי לפתור בעיות אלו, במערכות הפעלה של UNIX ו-LINUX אפשר ליצור קובץ אחד בשם Makefile, שבו יש את כל הפקודות שנרצה להריץ בטרמינל הקשורים לקימפול ולהרצה של הפרויקט שלנו. בתוך ה-Makefile נגדיר "חוק" עבור כל קובץ שנרצה ליצור, המכיל את התלות שלו בקבצים האחרים, ואת פקודות הקימפול שלו. כמו כן, נוכל להגדיר חוקים שימושיים נוספים. כעת בכל פעם שנרצה להריץ חוקים אלו, פשוט נצטרך להפעיל קובץ זה באמצעות פקודות make כמו שנלמד בהמשך.

פקודות make בטרמינל מחפשות אוטומטית בתיקייה שבה אנו נמצאים קובץ בשם "Makefile" ללא סיומת, ומריצות את החוק המתאים שכתוב בו. אם כתבנו קובץ בשם אחר, לדוגמא MyMakefile, נוכל להריץ פקודות make בו באופן הבא:

Make -f MyMakefile

## חוקים

קובץ Makefile מורכב מ"חוקים" (Rules). הצורה הכללית של חוק היא:

target [target...] : [dependent ....]

[command ...]

מה שמופיע בסוגריים המרובעות אופציונלי. מה שיש אחריו נקודות משמע אחד או יותר. חוק חייב להתחיל בתחילת שורה בלי רווחים מקדימים, אמנם בשורות הבאות הרווח (tab) הוא הכרחי.

**target** - זהו שם החוק, בדרך כלל שם הקובץ שאותו אנו רוצים ליצור לאחר הקומפילציה, אך ישמש גם להפעלת חוקים נוספים.

**dependent** - זהו הקובץ או הקבצים שקובץ המטרה שלנו תלוי בהם. אפשרות זו אופציונאלית.

**command** - אלו הפקודות שרוצים להריץ בהפעלת החוק. הפקודות הם פקודות רגילות למערכת ההפעלה כמו שמריצים בטרמינל.

דוגמא: אנו יוצרים קובץ hello שתלוי בקבצים main.o, factorial.o, hello.o ומקמפלים אותו.

hello: main.o factorial.o hello.o

$(CC) main.o factorial.o hello.o -o hello

אמנם גם כל הקבצים ש-hello תלוי בהם גם כן תלויים בקבצים אחרים, שהרי גם קובץ .o הוא תוצאה של קימפול. לכן עבור כל קובץ .o נצטרך להוסיף חוקים מהצורה הבאה, לדוגמא:

main.o: main.cpp functions.h

$(CC) -c main.cpp

factorial.o: factorial.cpp functions.h

$(CC) -c factorial.cpp

hello.o: hello.cpp functions.h

$(CC) -c hello.cpp

## Macros

Macros הם סוג של משתנים שניתן להשתמש בהם בקובץ Makefile.

### משתנים מיוחדים

$@ - זהו ה-target, השם של הקובץ שייווצר בתהליך הקימפול.

$? - זהו ה-dependents, השם של כל הקבצים שה-target תלוי בהם. שקול גם לסימון $@.cpp.

hello: main.cpp hello.cpp factorial.cpp

$(CXX) $(CXXFLAGS) **$?** $(LDFLAGS) -o **$@**

### משתנים רגילים

משתנים נוספים קבועים שלכל אחד יש תפקיד שונה וערכים מסוימים שהוא יכול לקבל. משתמשים במשתנים אלו בצורה של VARIABLE = VALUE. שני סוגי המשתנים הם:

* משתנים שהם שמות של תוכנות. לדוגמא CXX - תוכנה לקימפול קבצי cpp.
* משתנים שהם ארגומנטים שמתקבלים אל תוכנה. לדוגמא CXXFLAGS - דגלים לקומפיילר.

ניתן לראות את כל המשתנים <https://www.tutorialspoint.com/makefile/makefile_macros.htm>.

## פקודות make

כדי להריץ חוק כלשהו, נניח ששמו hello, נשתמש בפקודה make hello. משמעות פקודה זו היא תמצא את החוק hello, אם הקובץ hello לא קיים, או שהוא ישן יותר מאחד הקבצים שהוא תלוי בהם, אז בצע את הפקודות בשורה למטה. אמנם, אם אחד מהקבצים ש-hello תלוי בהם לא קיים בתיקייה, אזי יופעל רקורסיבית החוק של אותו קובץ. לדוגמא, עבור קובץ ה-Makefile המכיל את החוקים למטה, במידה ו-main.o ו-sum.o לא קיימים, הפקודה make sum תריץ רקורסיבית את החוקים של main.o ואת sum.o ורק לאחר מכן את השורה השנייה בחוק של sum. החוק של clean לא יבוצע.

sum: main.o sum.o

g++ –o sum main.o sum.o

main.o: main.cpp sum.h

g++ –c main.cpp

sum.o: sum.cpp sum.h

g++ –c sum.cpp

clean:

rm sum main.o sum.o

אם נריץ שוב make sum נקבל הודעה האומרת ש-sum כבר מעודכן ולא צריך ליצור אותו שוב. make בודק את זמן העדכון האחרון של קובץ המטרה, ואם הוא מאוחר יותר מזמן העדכון האחרון של קבצי המקור, הוא חוסך את פקודת הקימפול, וכך חוסך לנו זמן.

## make all

בתחילת קובץ ה-Makefile, בדרך כלל יהיה חוק לבניית מטרה בשם all, והוא תלוי בקובץ הראשי שאותו אנו רוצים להריץ. קובץ ראשי זה כמובן תלוי בכל שאר הקבצים בתוכנית שלנו. בחוק זה נרצה להריץ את הקובץ הראשי. לדוגמא:

all: a.out

./a.out

משמעות הכלל הזה היא בדיוק כמו משמעות הכללים הקודמים שלמדנו: "בדוק אם קיים קובץ בשם all ואם הוא מעודכן. אם הוא לא מעודכן הרץ את הפקודה בשורה הבאה. אך אם הקבצים בהם הוא תלוי לא קיימים צור אותם". אלא שכאן יש הבדל אחד קטן, הקובץ all אף פעם לא קיים כי אנחנו לא יוצרים אותו. ולכן הפקודה make all בעצם אומרת להריץ את קובץ ההרצה הראשי, ואם לא קיים יש להפעיל את כל החוקים ב-Makefile ואז להריץ אותו.

### make ללא פרמטרים

כשמריצים make בלי פרמטרים, הוא תמיד מנסה להפעיל את החוק הראשון בקובץ. וכמו שכבר אמרנו, בדרך כלל זה החוק all. לכן, כשכותבים רק make, הוא יעדכן ויריץ את התוכנה שלנו עם כל העדכונים האחרונים. בדיוק מה שאנחנו רוצים.

## make clean

ב-Makefile יש עוד חוק המתייחס למטרה שאף פעם לא נוצרת. בדרך כלל נרצה שבקובץ Makefile יהיה חוק שבעזרתו נוכל למחוק את כל קבצי ה-.o וקבצים נוספים. לכן מקובל ליצור חוק בשם clean שמטרתו למחוק קבצים אלו בעזרת הפקודה rm -f [files], שנכתוב בחוק זה. מפעילים את החוק על ידי הפקודה make clean. לדוגמא, כדי למחוק את כל קבצי .o ואת קובץ ההרצה הישן, נרשום את החוק:

rm -f \*.o a.out

## PHONY

כמו שאמרנו לעיל, נרצה לייצר משימות שלא תלויות בקבצים, כמו all ו-clean. אך מה יקרה כאשר בתיקייה שלנו כן יש קובץ בשמות אלו? במקרה זה אם קבצים אלו לא עודכנו מאז הפעם האחרונה שהפעלנו את המשימה, אזי כשנפעיל שוב את המשימה make יגיד לנו שהוא כבר מעודכן ולא יבצע אותה. כדי להימנע מכך נגדיר את כל המשימות מסוג זה כ-PHONY בדרך הבאה:

**.PHONY: clean**

clean:

rm -f \*.o