# 10. Tétel

Ismertesse a Makefile szintaktikai elemeit!

## GNU Compiler Collection

* Fajtái:
  + **GCC** – C
  + **G++** – c++, Objective C, Fortran
  + **GCJ** – Java
  + **GNAT** – Ada
* gyakran használt paraméterek
  + –o fájlnév

–c

* + –D definíció=x : definíció makró definiálása x-el
  + –I könyvtárnév: új header könyvtár elérési útvonalának megadása
  + –L könyvtárnév: source könyvtár hozzáadása
  + –static: default dinamikus linkelés helyett statikus használata
  + hibakeresés:
    - -g: szabványos hibainfók
    - -ggdb: olyan infókat helyezzen el a programban, amit csak a gdb értelmez
    - -ggdb N: debug info szint:
      * 0: nincs debug info
      * 1. default
      * 2: extra infók
  + –O, -O N: optimalizáció (szintek 0-3 között, 2 default)
  + –Wall: az összes warningot dobja consolera

## Make

### Általános szintaktika

* új sorban folytatása az előzőnek: \
  + mert a make minden sort egyesével hajt végre egy-egy subshellben 🡪 jelölni kell, ha még nincs vége a sornak
* hívás
  + make: első rule
  + make rule: argumentumban megadott rule
* Hamis tárgy: .PHONY: asdf
  + hogy a make ne értelmezze végrehajtandó szabályként az megadott kulcsszavakat
  + ilyen pl. a clean, hogy ha van clean fájlunk
* zárójelezés: () és {} egyenértékű

### Megjegyzések

* # karakterrel kell kezdődnie
* ha nem egy rule-ban van, akkor a make figyelmen kívül hagyja

### Explicit rules

* meghatározza, hogy mikor és hogyan kell újrafordítani egy vagy több állományt
* target : dependency list
* a targetet akkor kell újrafordítani, ha
  + a target nem létezik
  + a dependency list-ben valami módosult (az időbéjeg későbbi, mint a targeté)
* commands, receipt
  + TAB-al kell kezdeni, nem szóközzel!!

### Változódefiníciók

* VÁLTOZÓ = ÉRTÉK
* rá való hivatkozás: $(VÁLTOZÓ) vagy $({VÁLTOZÓ}
* a változó nevében nem lehet spec karakter (:, #, =), üres mező
  + gyakorlatban csak betűk számok és \_ lehet benne
* saját változónevek: inkább kisbetű, nagybetűket rendszerváltozóknak meghagyják
* olyan, mint a #define (bármi belekerülhet), más objektumok, rövid makrók stb.
* append, behelyettesítés: NEM SZEKVENCIÁLIS
* speciális esetek:
  + egyszerű kiértékelés helyben: := használatával
  + += hozzáfűzés
  + ?= : feltételes értékadás – a változónak csak akkor adunk értéket, ha még nincs definiálva
    - a = egy
    - a ?= kettő
    - eredmény: egy
  + többsoros változók:
    - define változó =
    - muhaha
    - hehe
    - endef
  + változó törlése: undefine változó
* változó hivatkozások
  + a változóra hivatkozás során az értékét módosítva helyettesítsünk be az adott helyre
  + konverziós szabály (az értéket szavanként kezeljük)
  + pl.:

srcs = elso.c masodik.c

objs := $(srcs:.c=.o)

* + számított változónevek:

src\_1 := elso.c

src\_2 = masodik.c

objs := $(src\_$(a):.c=.o)

az a változó érékétől függően vagy az src\_1 vagy az src\_2 értékéből állítjuk elő az objs értékét, majd c-ről ó-ra cseréljük a kiterjesztését

* automatikus változók
  + $@ - target
  + $< – dependency list első eleme
  + $? – teljes dependency list-ből azok, akik módosítva lettek
  + $^ – teljes dependency list (ha valami többször szerepel, csak egyszer helyettesíti be)
  + $+ – teljes dependency list
  + $\* – target kiterjesztés nélkül

### Többszörös target

Többszörös cél:

debug\_flags := -g3 –o0

release\_flags := -g0 –o3

prgs := debug release

all : $(prgs)

\*PHONY: all

$(prgs) : hello.c

gcc $($(@)\_flags) –o $@ $<

A make parancs kiadása után két fordítás történik. Létrejön egy debug és egy release állomány eltérő paraméterezéssel

gcc –g3 –o0 –o debug hello.c

gcc –g0 –o3 –o release hello.c

* + targetek listáját is meg lehet adni a target mezőben
  + külön-külön végrehajtódik a szabály minden egyes targetre
  + a recept lehet hasonló, de a végrehajtott parancsok különbözőek: automatikus változók

### Minta szabályok

* + s statikus szabályokkal ellentétben ez egy alapértelmezett szabályt ad. De ha a makefileban van más specifikus szabály, akkor az fog végrehajtódni
* TAGET : TARGET-TEMPLATE : DEPENDENCY TEMPLATE
  + $(objs) : $.o: $.c
  + iterál
* TARGET-TEMPLATE : DEP-TEMPLATE
  + $.o : $.c
    - minden \*.o-ra a következő szabály hajtódjon végre

### Klasszikus ragozási szabályok

* .c .o: \ receipt \ .SUFFIXES: .c .o
  + ugyanazt csinálja, mint az előző
  + .SUFFIXES: speciális tárgy, ami után fel kell sorolnunk párinként melyik kiterjesztésből áll elő egy másik
  + nem használjuk

### Speciális tárgyak

* + .PHONY – hamis célok kivételei
  + .SUFFIXES – a dependency list-ben állománynév kiterjesztések vannak, amelyeket a make a ragozási szabályok keresésénél használ fel
  + .SILENT – a dependency listbe írt parancsok eredményeit nem teszi ki a kimenetre
  + .ONESHELL – a parancsok ne subshellekben hívódjanak

### Direktívák

* + a make „prerocesszora”
  + a következő műveleteket definiálhatják
    - más makefile beolvasása
      * include fáljnév
    - a makefile bizonyos részeinek használának engedélyezése, tiltása
    - ifeq ($(változó1) , $(változó2))
      * else
      * endif

## Make alternatívák

Komplexebb feladatok ellátására, sokszor make alapú programok

### Autotools

* make + extra szabálylista + függőség ellenőrzése
* célja a forráskódok hordozhatósága platformok között
* fő részei: Autoconf, Automake, Libtool
* a fordításhoz nincs szükség a teljes Autotoolshoz, elég ha a configure script megvan, ami definiálja a műveleteket

### CMake

* fordítás
* VS projekt állomány előállítása
* gondoskodik a szükséges állományok legenerálásáról is

### qmake

* Qt része, azok fordítását támogatja
* makefile-t generál

### SCons

* szoftver konstrukciós eszköz
* függőségek vizsgálata
* platform adaptáció támogatása
* Python alapú