Nội dung

**Define**

* + Make tool
  + Makefile

**Basic Syntax**

* + Make tool
  + Makefile
    - A simple Makefile

**Rule**

* + - * Variable
      * Syntax of Conditionals
      * Implicit Rules
      * Phony Target
      * Including Other Makefiles

**Make Tips & Tricks**

**Refer**

### Make

* Automatically determines which pieces of a large program need to be recompiled.
* Give rules how to transform them.
* Produce output files from several input files.
* Can be used for any kind of “compilation task”.
  + Preparing LATEX documents.
  + Transforming images,…

Several variants exist:

* + - GNU Make (covered in this lecture).
    - Microsoft make.
    - BSD make.

### Makefile

* A **makefile** is a file (by default named "Makefile") containing a set of directives used by a **make** tool to generate a target/goal.

The ***makefile*** also tells **make** how to compile and link a program

|  |
| --- |
| # This is a very simple makefile  MCU:  echo “Welcome to MCU team” |

### Make Tool: Syntax Overview

***For example:***

* A file **Makefile** with this content:
* Run the file by typing **make**, the output will be:

$ make

echo “Welcome to MCU team”

Welcome to MCU team

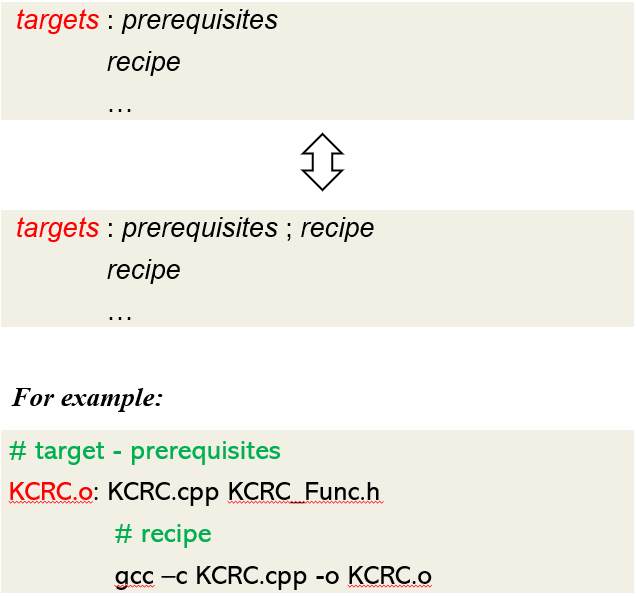
$ make MCU

echo “Welcome to MCU team”

Welcome to MCU team

### Basic Syntax of Makefiles

* A *makefile* consists of a set of **targets**, **prerequisites**and **recipe**.
* The **targets** are file names, separated by spaces, to be created/updated.
* **target** depends upon a set of source files or even other **targets** described in **prerequisites**.
* The **recipe** of a rule consists of one or more **shell** command lines to be executed, one at a time, in the order they appear.
* **Note**: Each command in the **recipe** must be on lines that start with a **TAB** character. **Space** issue (\*\*\*\*) errors.



### A Simple Makefile

A picture containing shape

Description automatically generated

* Suppose we have a small project containing.
  + Two source file: **KCRC.cpp** and **KCRC\_Func.cpp**
  + Header files: **KCRC.h**, **KCRC\_Func.h**, **KCRC\_AgentController.h** and **kcrc\_regif.h** which included by both .cpp files.
* To build: We compile all .cpp files to .o files and then link the .o files together.
* When we develop (edit .cpp and .h files).
  + We need to rebuild the .o files affected by the changes, and finally the binary.
* Writing the appropriate compiler invocations by hand all the time is cumbersome.
* **KCRC\_Func.h** and **KCRC.cpp** are prerequisites for **KCRC.o**
  + So **KCRC.o** needs to be rebuilt when one of those are changed.
* Our example:
* According to the rules. Make constructs a dependency graph.
* This graph needs to be acyclic (DAG).
* When processing the Makefile, Make traverses the graph from leaves to root.
* If the modification date of a child is newer than the node’s, the node needs to be redone.

Dependencies in our example:

Graphical user interface, text, application

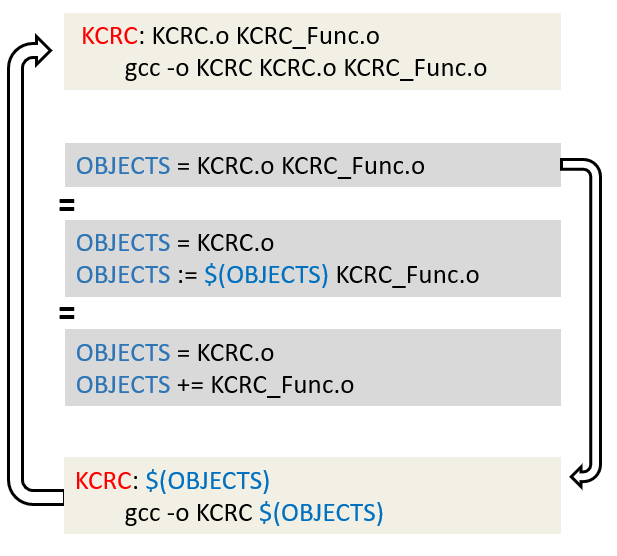
Description automatically generated

Diagram

Description automatically generated

### Rule – Variables (1/2)

* In the above example, most target and prerequisite values are hard-coded, but in real projects, these are replaced with variables and patterns.
  + The simplest way to define a variable in a makefile is to use the **’=‘** operator.
* We can also use: **‘:=‘**, **‘+=‘**, **‘?=‘**, **‘!=‘**
  + Variables defined with **‘=’** are variables recursively expanded.
    - The value of a variables recursively expanded is installed verbatim; if it contains references to other variables, these references are expanded whenever this variable is substituted.
  + Variables defined with **‘:=’** or **‘::=’** are simply expanded variables.
    - The value of a simply expanded variable is scanned once and for all, expanding any references to other variables and functions, when the variable is defined.
  + Variables defined with **‘+=’** are useful to add more text to the value of a variable already defined.
  + Variables defined with **‘?=’** are variables to be set to a value only if it’s not already set.
  + The shell assignment operator **‘!=’** can be used to execute a shell script and set a variable to its output.



**Rule – Variable (2/2)**

Graphical user interface, text, application, chat or text message

Description automatically generated