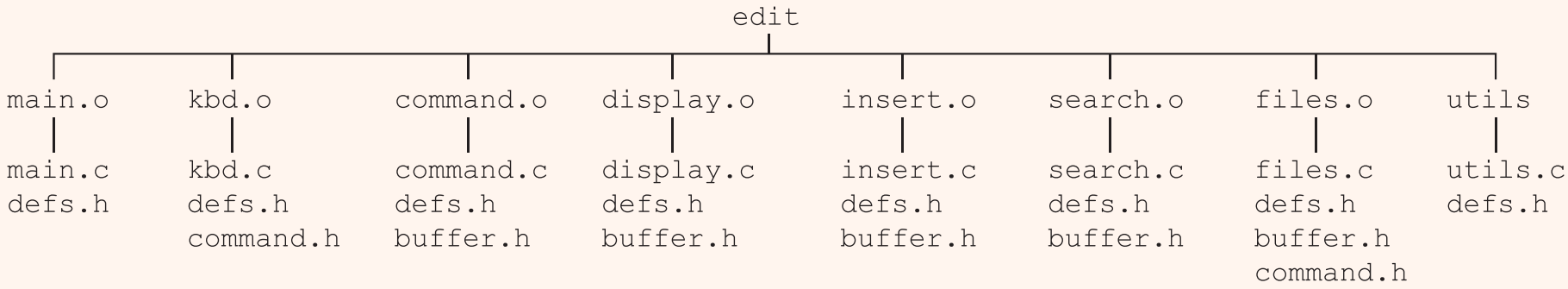
GNU *Make*[1]

- The *make* utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.
- *Make* can be used with any programming language whose compiler can be run with a shell command.
- *Make* is not limited to programs. It can be used to describe any task where some files must be updated automatically from others whenever the others change.
- We will use *make* to automatically generate executable images (*.hex files) for downloading from our *.c and *.h files.

[1] material adapted from "GNU Make" by Richard Stallman

GNU *Make*

- Here is a dependency graph for a program called edit.



- *Make* works by building a source file dependency graph and checking file creation times to determine which files are now out of date and recompiling only as needed.
- For example, if `command.h` was changed, its timestamp would be newer than the `*.c` and `*.h` files that depend on it.
- Thus *make* would recompile `kbd.c` and `files.c` and relink all the object files to produce edit. No other files would be compiled as there is no need.

GNU *Make*

- *Make* is controlled by a makefile which describes dependencies and actions to take to rebuild the executable image.
- The makefile is usually called either `Makefile` or `makefile`.
- Advantages of using a makefile to build a project:
  - The makefile documents exactly how to rebuild the executable
  - It reduces rebuilding a complex project to a single command
  - It can be used to build the other files you need: .eeprom, .lst, .hex...
  - It is very general purpose: VHDL, Verilog, LaTeX, etc.
  - One makefile can be used reused for other projects with minor edits

GNU *Make*

- *make* looks for a makefile in the current directory. The makefile describes the dependencies and actions using makefile *rules*
- A *rule* is formed as follows:

```
target          :  prerequisites
<hard tab>         commands
```

- *target*: usually the name of a file generated by the command below. It can also be the name of an action to carry out. (phony target)
- *prerequisites*: file(s) that used as input to create the target. A target usually depends on several files.
- *commands* : usually shell command(s) used to create the target. Can also specify commands for a target that does not depend on any prerequisites. (e.g.; clean)

GNU *Make*

- An example rule:

```
sr.o   :   sr.c
           avr-gcc -g -c -Wall -O2 -mmcu=atmega128 -o sr.o sr.c
```

- This rule tells make that:
  - `sr.o` is dependent on `sr.c`
  - if `sr.c` is newer than sr.o, recreate `sr.o` by executing avr-gcc
- A rule then, explains how and when to remake certain files which are the targets of the particular rule. *make* carries out the commands on the prerequisites to create or update the target.

GNU *Make*

- *make* does its work in two phases:
  1. Include needed files, initialize variables, rules, build dependency graph
  2. Determine which targets to rebuild, and invoke commands to do so
- The order of the rules is insignificant, except for determining the default goal.
- Rules tell make when targets are out of date and how to update them if necessary.
- Targets are out of date if they do not exist or if they are older than any of its prerequisities.

GNU *Make*

- A simple makefile for `edit`

```
edit       : main.o kbd.o command.o display.o insert.o search.o files.
             utils.o
    cc -o edit main.o kbd.o command.o display.o insert.o search.o\
             files.o utils.o
files.o    : utils.o
main.o     : main.c defs.h
             cc -c main.c
kbd.o      : kbd.c defs.h command.h
             cc -c kbd.c
command.o  : command.c defs.h command.h
             cc -c command.c
display.o  : display.c defs.h buffer.h
             cc -c display.c
insert.o   : insert.c defs.h buffer.h
             cc -c insert.c
search.o   : search.c defs.h buffer.h
             cc -c search.c
files.o    : files.c defs.h buffer.h command.h
             cc -c files.c
utils.o    : utils.c defs.h
             cc     utils.c
clean :
             rm edit main.o kbd.o command.o display.o insert.o search.
             utils.o
```

GNU *Make*

- A simple make file for sr.c including variables

```
Listing goes here
```

GNU *Make*

- We can make the makefile more general purpose with some new rules.
  - *Implicit rules* - *make* implicitly knows how to make some files. Its has implicit rules for updating `*.o` files from a corresponding `*.c` file using gcc. We can thus omit commands for making `*.o` files from the object file rules.
  - *Pattern rules* - You can define an *implicit* rule by writing a pattern rule. Pattern rules contain the character %. The target is considered a pattern for matching names.
    The pattern rule:
    ```
    %.o   :   %.c
              command
    ```
    Says to create any file ending with `.o` from a file ending with `.c` execute `command`.

GNU *Make*

- Automatic variables
  - These are variables that are computed afresh for each rule that is executed based on the target and prerequisites of the rule.
  - The scope of automatic variables is thus very limited. They only have values within the command script. They cannot be used in the target or prerequisite lists.
  - Some automatic variables we will use:

    `$@` The file name of the target of the rule

    `$<` The name of the first prerequisite

    `$?` The names of all the prerequisites newer than the target

    `$^` The names of all the prerequisites

GNU *Make*

```
Big make file goes here
```

GNU *Make*

- Generally, translational units are turned into object files. But header files are not standalone translational units, *make* has no way of knowing that it needs to rebuild a `.c` file when a `.h` file changes.
- gcc is able to read your source file and generate a list of prerequisite files for it with the -MM switch.
- We can take this output and create another makefile from it using some shell commands. There will be one makefile for each source file. Each makefile is then included in our top level makefile so that the dependencies are taken care of.
- Here is the makefile code for this:

```
%.d: %.c
        @set -e; rm -f $@; \
        $(CC) -M $(CFLAGS) $< > $@.$$$$; \
        sed 's,\($*\)\.o[ :]*,\1.o $@ :,g' < $@.$$$$ > $@; \
        rm -f $@.$$$$

-include ($SRCS:.c =.d)
```