Ch. 11: C Programming Tools

Compiling using gcc:

```
% gcc reverse.c
```
produces executable in a.out

```
% gcc -o reverse reverse.c
```

Multi-module program:

```
types.h supps.h, supps.c parts.h, parts.c sp.h, sp.c
db.h, db.c (this has the main() function)
```

Compile these programs separately;

```
% gcc -c supps.c
% gcc -c parts.c
% gcc -c sp.c
% gcc -c db.c
```

These commands produce supps.o, parts.o, sp.o, db.o

```
% gcc db.0 supps.o parts.o sp.o -o db
```

```
Unix File-Dependency System: make

% make -f fileName

Makefiles: consist of a list of interdependency rules of the form:

  targetList:dependencyList
             commandList

where targetList is a list of target files
      dependencyList is a list of files on which the files in targetList
                 depend on
      commandList is a list of zero or more commands separated by new lines
                 that reconstruct the target files from the dependency files.

      NOTE: Each line in commandList MUST start with a tab
            Rules must be separated by at least one line
```

Example:

```
db: db.o supps.o parts.o sp.o
      gcc db.o supps.o parts.o sp.o -o db

db.o: db.c types.h db.h supps.h parts.h sp.h
      gcc -c db.c

supps.o: supps.c types.h supps.h
      gcc -c supps.c

parts.o: parts.c types.h parts.h
      gcc -c parts.c

sp.o: sp.c types.h sp.h
      gcc -c sp.c
```

Order of Make rules:

The order of make rules is important. The make utility starts from the first rule and creates a tree with target files at the root and the dependency files as children.

```
                        db

     db.o                                supps.o       parts.o          sp.o

db.c types.h supps.h parts.h sp.h   supps.c types.h supps.h

                                       parts.c types.h parts.h

                                                           sp.c types.h sp.h
```

The make utility then works up the tree from the leaf nodes to the root node, looking to see if the last modification time of each node is more recent than the last modification time of the immediate parent node. If so, the associated parent's rule is executed.

Make Rules of the form:

```
xxx.o: reverse.c reverse.h
      gcc -c xxx.c
```

where xxx varies from each rule

Pre-defined rule:

```
.c.o:
      /bin/cc -c -O $<
```

So makefile can be simplified as:

```
db: db.o supps.o parts.o sp.o
      gcc db.o supps.o parts.o sp.o -o db

db.o: db.c types.h db.h supps.h parts.h sp.h
supps.o: supps.c types.h supps.h
parts.o: parts.c types.h parts.h
sp.o: sp.c types.h sp.h
```

Further simplification (leave out .c files in dependency rules)

```
db: db.o supps.o parts.o sp.o
     gcc db.o supps.o parts.o sp.o -o db

db.o: types.h db.h supps.h parts.h sp.h
supps.o: types.h supps.h
parts.o: types.h parts.h
sp.o: types.h sp.h
```

```
touch -c {fileName}+
```

updates the last modification and access times of the named files to the current time. By default, if a specified file does'nt exist it is created with zero size. To prevent this default, use -c option.

use touch to force make to recompile files.