Organising larger programs

The C-Preprocessor

C-Preprocessor called before compiler
Does syntactic transformations on code

Most important commands:

- `#include <filename>`: insert file into source code
- `#define <name> <text>`: replace `<name>` by `<text>`.
  Macros may have arguments:
  ```
   #define MAX(A,B) ((A) > (B)) ? (A) : (B))
  ```
- `#if <expression>` include following text only if `<expression>` is true
  Expression must contain only constants
  `#endif` and `#else` delimit the scope
- `#defined(<name>)` returns 1 if `<name>` has been declared by `#define <name>` and 0 otherwise.

Handling larger programs

Program may be split amongst several files
compile each file separately via

```
gcc -Wall -Werror -c <filename>.c
```

and link them all together with

```
gcc -Wall -Werror -o <filename> <filename1>.o <filename2>.o ..
```

If function or variable defined in one file and used in other one, need to have `extern` declaration of these functions or variables in other file

Creating libraries

Often used routines may be assembled into library
Creation of library:

- Create object files as usual
- Create library via

```
gcc -Wall -Werror -shared -fPIC -o lib<library>.so <filename1>.o ..
```

Managing directories

Need to tell compiler where include-files and libraries are located

- Include-files
  C-compiler searches `/usr/local/include`, `/usr/include` and compiler-specific directories for include-files
  Any other directories must be specified with `-I`-option
- Libraries
  C-compiler searches `/lib` and `/usr/lib` plus compiler-specific directories for libraries
  Any other directories must be specified with `-L`-option
- Option `-l` specifies all libraries which are to be used.

Shared libraries need to be loaded by executable as well

$\Rightarrow$ need to tell executable where they are

Again, standard locations are `/lib` and `/usr/lib`

Have environment variable `LD_LIBRARY_PATH` to specify additional directories

Set by command

`export LD_LIBRARY_PATH=<Directory1>` for one additional directory, and

`export LD_LIBRARY_PATH=<Directory1>:<Directory2> ...` for several directories

Correct sequence for compiling example:
First, the library:

```
gcc -Wall -Werror -fPIC -c input.c
c -Wall -Werror -g  -fPIC -shared  -o libinput.so input.o
```

Now the program itself:

```
gcc  -Wall -Werror -g  -I../include -c program.c
gcc  -Wall -Werror -g  -I../include -c output.c
gcc -Wall -Werror -g   -L../lib   -o program output.o program.o
```

Automated Compiling

Have program called `make` which can control compilation of large program systems
Actions of `make` controlled by so-called `Makefile`
Structure of simple Makefile :

- Starts with declarations (assignment of values to variables)
- Then have list of targets and commands which re-create the target
  Need to have TAB-character at beginning of line containing the command!
- Can call `make` in directory `<dir>` via
  `make -C <dir>`

- Such a list of targets consists of one line containing target and dependencies
- dependencies are files which need to be present and newer than the target
- commands will be executed if target needs to be re-generated variables may be used

Conventions:

- Have target `all` which makes everything in this directory normally first target
- Have also target `clean` which removes all targets and temporary files created