



7. Structuring Code

- Separate compilation
- Header files
- Examples
- Makefiles



Separate compilation

- So far, when using *gcc* we have translated a source program to an executable code file.
- Actually, this has involved 4 separate programs:
 - *cpp* to pre-process the source file
 - *gcc* to translate C to assembly language
 - *as* to translate assembly language to a relocatable binary *object code* file
 - *ld* to link your code with the library code and build an executable code file



Separate compilation (cont'd)

- If we pass a `-c` flag to `gcc` it stops after the third step
- Output is a *relocatable object module*
 - Binary code plus:
 - a table of all locations that need to be changed when the code is moved in memory during linking
 - a table of all locations that reference externally defined symbols (e.g. calls to `printf`)
 - a table of all locations that define externally-referenceable symbols (e.g. callable functions)



Example

- Suppose we have a source code file *silly.c*:

```
int myVeryUsefulFunction(int arg)
{
    return arg;
}
```

- Then could compile with:

```
gcc -c $CFLAGS silly.c -o silly.o
```

- Now have an *object file* called *silly.o* which contains a function that can be used by other programs.



Using that object file

- Can use *silly.o* from program *prog.c* as follows:

```
int myVeryUsefulFunction (int arg); // So compiler knows its type

int main(void)
{
    int n = 10;
    printf("%d\n", myVeryUsefulFunction(n)); // Prints 10!
}
```

- Compile *prog.c* with:

```
- gcc $CFLAGS prog.c silly.o -o prog
```



Gets passed straight through to the linker



Header files

- Don't want to have to declare the signature of all external functions
- So instead, put all signatures of an external .o file in a separate C file with the extension *.h*
 - Called a *header file*
- For example, *silly.h* would contain just the line

```
int myVeryUsefulFunction(int arg);
```

- *prog.c* would then be

```
#include "silly.h"
int main(void) {
    int n = 10;
    printf("%d\n", myVeryUsefulFunction(n)); // Prints 10!
}
```



Lab 5 example

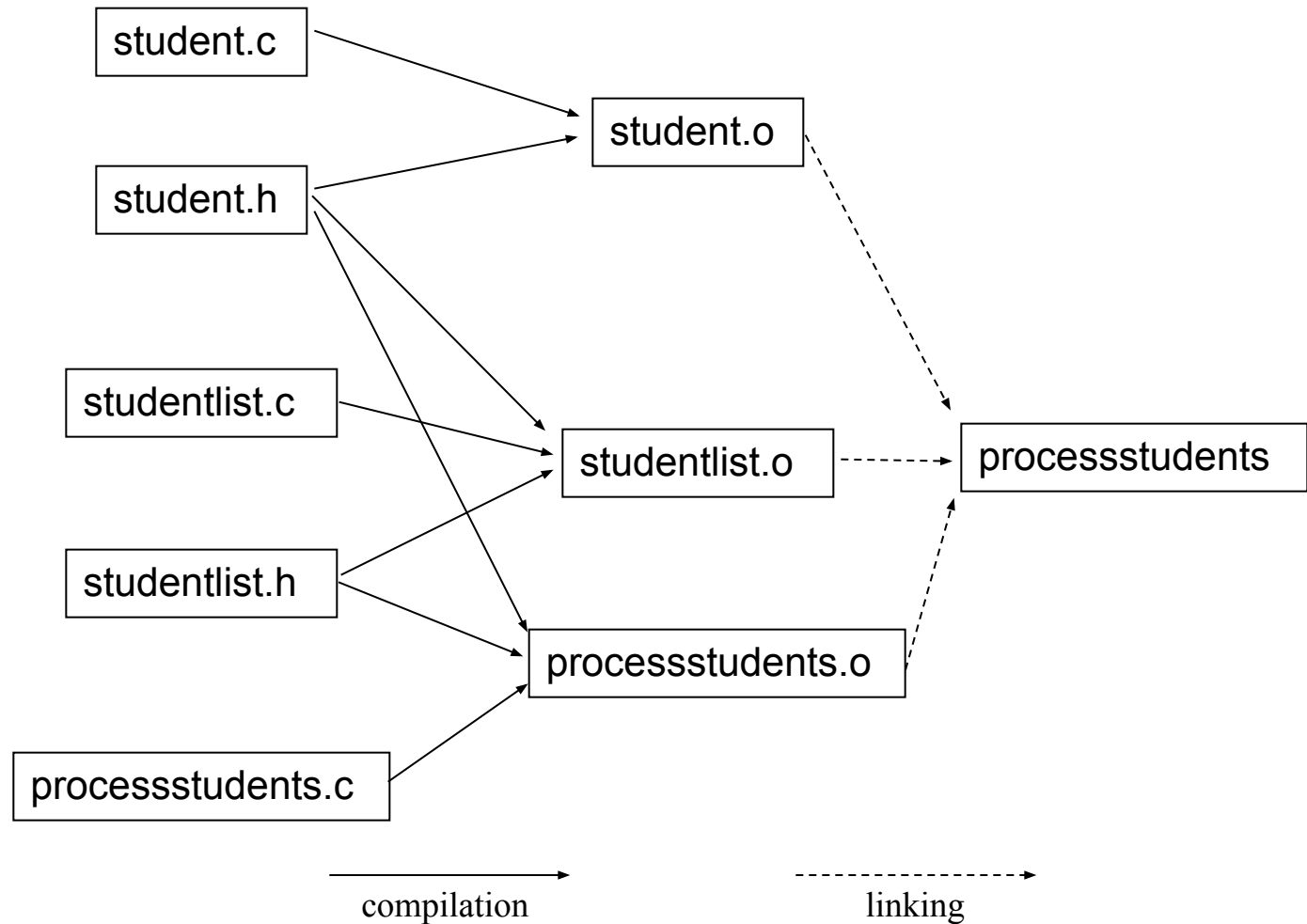
Breaks the code of *structexample3.c* into 3 modules:

1. The *Student* module: *student.c*, *student.h*
 - Typedef for *Student*; function *newStudent*, *readOneStudent*; student pool
2. The *StudentList* module: *studentlist.c*, *studentlist.h*
 - Functions *readStudents*, *addStudent*
 - These are methods of a (hypothetical) *student list*
3. The main program, *processStudents.c*
 - Begins:

```
#include "student.h"
#include "studentlist.h"
```
 - Reads students from file, prints the list



Dependency graph





Makefiles

- To reap full benefit of separate compilation, want to recompile only the changed modules, then relink.
- Done by *make*
 - Takes a *make file* as a parameter
 - Default name is *Makefile* or *makefile*
 - A make file is a textual description of the dependency graph plus the commands that make each target
 - Consists of a set of node descriptions of form

```
targetNodeName: sourceNode1 sourceNode2 ...  
<tab>Rule to make target node if source nodes all made
```
 - Most rules can be omitted, as default rule to make a .o file is to compile the corresponding .c file



Example makefile (full)

Warning: all indents are with a single TAB char

```
# Definitions.
CFLAGS = -g -std=c99 -Wall -Werror

# Main target: create executable by linking the object files
processstudents: processstudents.o student.o studentlist.o
    gcc processstudents.o student.o studentlist.o -o processstudents

# Intermediate targets: create object files from C source codes.
processstudents.o: processstudents.c student.h studentlist.h
    gcc $(CFLAGS) -c processstudents.c

student.o: student.c student.h
    gcc $(CFLAGS) -c student.c

studentlist.o: studentlist.c student.h studentlist.h
    gcc $(CFLAGS) -c studentlist.c

# "Clean up" pseudo-target (used with the command "make clean")
Clean:
    rm *.o processstudents
```



Example makefile (minimised)

Warning: all indents are with a single TAB char

```
# Definitions.
CFLAGS = -g -std=c99 -Wall -Werror
OBJECTS = processstudents.o studentlist.o student.o

# Targets
processstudents: $(OBJECTS)
    $(CC) -o processstudents $(OBJECTS)

processstudents.o studentlist.o: studentlist.h

processstudents.o studentlist.o student.o: student.h

Clean:
    rm *.o processstudents
```