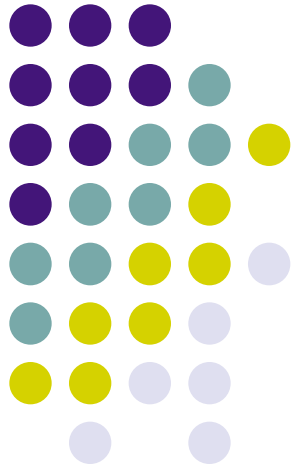
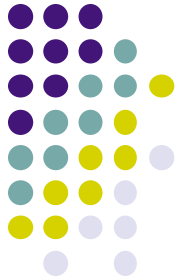
COMP20003 Algorithms and Data Structures Header Files and Makefiles

Nir Lipovetzky
Department of Computing and Information Systems
University of Melbourne
Semester 2

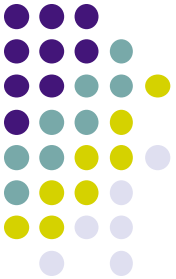


Multifile programs



- Multifile programs allow you to reuse code.
- For example:
  - Functions associated with a dictionary can be used over and over in different programs.
  - Can change underlying data structure for the same program, *e.g.* change from a list dictionary to a bst.

Header files



- Header files allow:
  - write a function prototype or definition once
  - then use it in different files.
- For example:

```
#define TRUE 1
#define FALSE 0
```

- or

```
int comp(char *, char *);
```

Header files

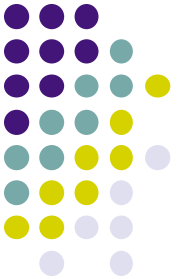
- To avoid retyping (and likely errors!), put the definitions in a header file, *e.g.*

  `header.h`

- And in your program file(s) include the header:

  `#include "header.h"`

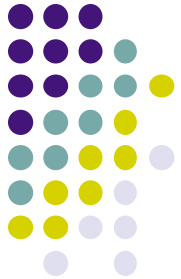
Including header files



```
#include <stdio.h>
#include "header.h"

int
main()
{
    /* code in here can use
      definitions and prototypes in
      header*/
}
```

Compiling multifile programs



- `gcc -o dict1 dict1.c bst1.c`
  - Prone to typing errors.
  - Recompiles everything from the ground up.
- Makefiles
  - Simplify the compilation command:
    - `make dict1`
  - Check what files have been changed, and only recompile them.

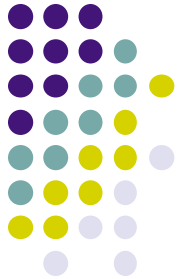
Makefile

```
dict1: dict1.o bst1.o
	gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
	gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
	gcc -c -Wall dict1.c
```

Makefile



target

```
dict1: dict1.o bst1.o
	gcc -o dict1 dict.o bst1.o
```

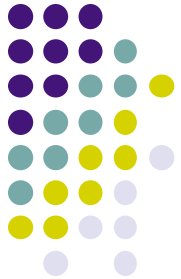
target

```
bst1.o: bst1.c bst1.h
	gcc -c -Wall bst1.c
```

target

```
dict1.o: dict1.c dict.h
	gcc -c -Wall dict1.c
```

Makefile



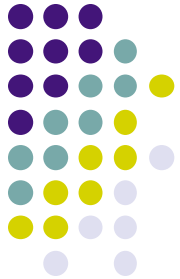
dependencies

```
dict1: dict1.o bst1.o
	gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
	gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
	gcc -c -Wall dict1.c
```

Makefile



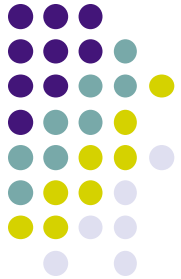
instructions

```
dict1: dict1.o bst1.o
    gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
    gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
    gcc -c -Wall dict1.c
```

Makefile



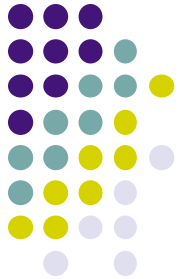
instructions

```
dict1: dict1.o bst1.o
    gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
    gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
    gcc -c -Wall dict1.c
```

Makefile



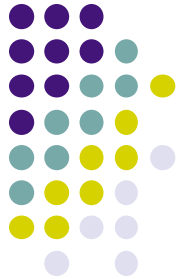
instructions

```
dict1: dict1.o bst1.o
    gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
    gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
    gcc -c -Wall dict1.c
```

Makefile



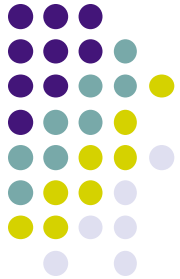
instructions

```
dict1: dict1.o bst1.o
    gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
    gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
    gcc -c -Wall dict1.c
```

Makefile



instructions

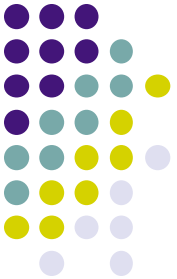
```
dict1: dict1.o bst1.o
	gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
	gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
	gcc -c -Wall dict1.c
```

tabs

Makefile



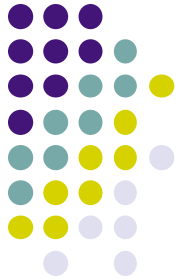
```
dict1: dict1.o bst1.o
	gcc -o dict1 dict.o bst1.o

bst1.o: bst1.c bst1.h
	gcc -c -Wall bst1.c

dict1.o: dict1.c dict.h
	gcc -c -Wall dict1.c

# comments start with hash
```

More on .o files, linkers, etc.



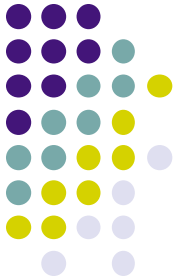
- http://www.lurklurk.org/linkers/linkers.html

David Drysdale, “Beginner’s Guide to Linkers”

Header files

- Contains
  - function declarations
  - macro definitions
  - shared among several source files.

Another example: `list.h`



```
#ifndef LISTH
#define LISTH
…
typedef struct node{
    data_t  data;
    node_t   *next;
} node_t;
     …
int search_sorted(list_t  *list,data_t value);
     …
#endif
```