

COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 14

More linked list - back to the ice-cream shop we go
Multi-file projects

LAST TIME...

- Deleting from a linked list
 - nowhere to delete (empty list)
 - at the head
 - at the tail
 - in the middle!

TODAY...

- Back to the ice-cream shop example - rehash and build on it (putting together a linked list example with inserting, traversing and deleting)
- Talk about using leakcheck command
- Multi-file projects

“

WHERE IS THE CODE?



Live lecture code can be found here:

[HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T1/LIVE/WEEK08/](https://cgi.cse.unsw.edu.au/~cs1511/22T1/LIVE/WEEK08/)

HARDER EXAMPLE

LET'S RETURN TO THE ICE-CREAM SHOP

I've changed my mind on the problem, and now we are going to run an ice-cream shop...

I have decided to run an ice-cream shop (no surprises here)

I want to create a program that can:

- 1) Take in all the flavours of ice cream that I have to offer by adding them to the end of the list
- 2) I want to be able to print out the flavours
- 3) I would then like to add the flavours in alphabetical order
- 4) Delete flavours as we finish them.

Starter code is provided

HARDER EXAMPLE

```
struct shop {  
    struct ice_cream *flavour;  
};
```

```
struct ice_cream {  
    int number;  
    char name[MAX_LENGTH];  
    struct ice_cream *next_flavour;  
};
```

shop



ice_cream



head of
ice_cream
nodes
stored in
flavour
pointer,
inside the
shop struct so
shop->flavour

BREAK TIME...

A prisoner is locked up in a tower with two doors. One of them leads to the exit, the other to the oubliettes. A guard is stationed in front of each door. One of them always tells the truth, the other lies systematically.

What one question must the prisoner ask just one of the guards to be certain of finding out which door leads to freedom?

MULTI FILE PROJECT

WHAT ARE THEY?

- Big programs are often spread out over multiple files. There are a number of benefits to this:
 - Improves readability (reduces length of program)
 - You can separate code by subject (modularity)
 - Modules can be written and tested separately
- So far we have already been using the multi-file capability. Every time we `#include`, we are actually borrowing code from other files
- We have been only including C standard libraries

MULTI FILE PROJECT

WHAT ARE THEY?

- You can also `#include` your own! (FUN!)
- This allows us to join projects together
- It also allows multiple people to work together on projects out in the real world
- We will also often produce code that we can then use again in other projects (that is all that the C standard libraries are - functions that are useful in multiple instances)

MULTI FILE PROJECT INCLUDES

.H FILE

**.C FILE (MAYBE
MULTIPLES)**

- In a multi file project we might have:
- (multiple) header file - this is the .h file that you have been using from standard libraries already
- (multiple) implementation file - this is a .c file, it implements what is in the header file.
- Each header file that you write, will have its own implementation file
- a main.c file - this is the entry to our program, we try and have as little code here as possible

header
file

#include " .h"

impleme
ntation
file

.c

HEADER FILE

#INCLUDE

"SOMETHING.H"

Typically contains:

- function prototypes for the functions that will be implemented in the implementation file
- comments that describe how the functions will be used
- #defines
- the file basically SHOWS the programmer all they need to know to use the code
- NO RUNNING CODE
- This is like a definition file

IMPLEMENT ATION FILE

SOMETHING.C

This is where you implement the functions that you have defined in your header file

IMPLEMENT ATION FILE

MAIN.C

This is where you call functions from that may exist in other modules.

AN EXAMPLE

A MATHS

- We will have three files:
 - header file - maths.h
 - #include "maths.h"
 - implementation file - maths.c
 - #include "maths.h"
 - main file - main.c
 - #include "maths.h"

```
*maths.h
1 // This is the header file for the maths module example
2 // The header file will contain:
3 //     - any #define
4 //     - function prototypes and any comments
5
6 #define PI 3.14
7
8 //Function prototype for a function that calculates
9 //square of a number
10 int square(int number);
11
12 //Function prototype for a function that calculates
13 //sum of two numbers
14 int sum(int number1, int number2);
15
```

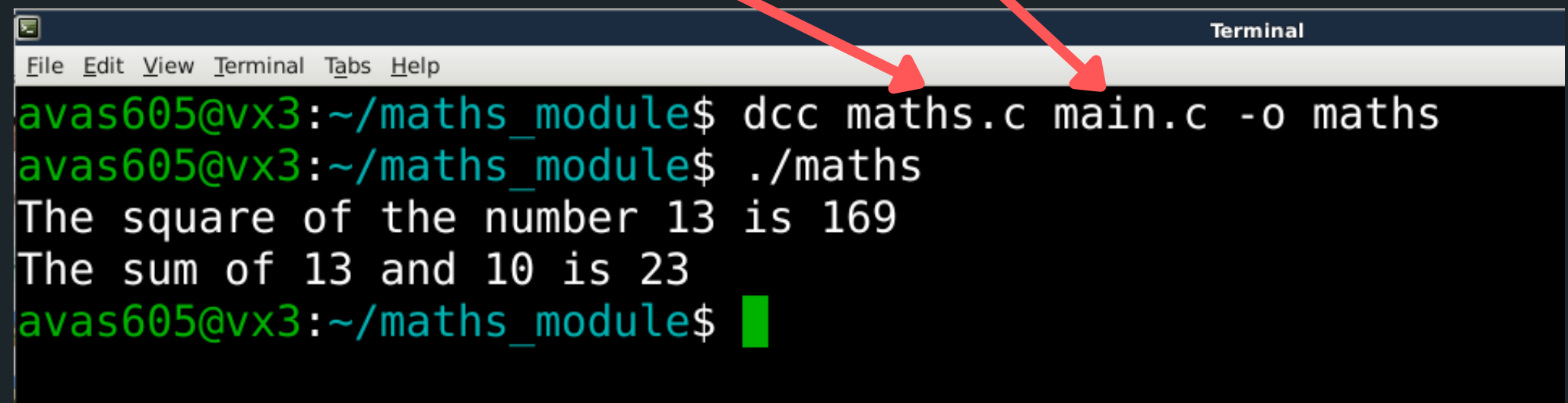
```
main.c
1 //This is the main file in our program
2 //This is where we drive the program from and where we
3 //make calls to our modules. We need to include the
4 //header file for each module that we want to use functions
5 //from
6
7 #include <stdio.h>
8 //Include the header file:
9 #include "maths.h"
10
11 int main (void) {
12     int number = 13;
13     int number2 = 10;
14
15     printf("The square of the number %d is %d\n", number, square
(number));
16     printf("The sum of %d and %d is %d\n", number, number2, sum
(number, number2));
17     return 0;
18 }
```

```
maths.c
1 //This is the implementation file of maths.h
2 //We defined two functions in the header file,
3 //and this is where we will implement these two
4 //functions
5
6 //Include your header file in the implementation file
7 //by using the below syntax
8
9 #include "maths.h"
10
11 int square(int number) {
12     return number * number;
13 }
14
15 int sum(int number1, int number2) {
16     return number1 + number2;
17 }
```

COMPILING A MULTI FILE

COMPILE ALL C FILES IN THE PROJECT

- To compile a multi file, you basically list any .c files you have in your project
 - In the case of our example, we have a maths.c and a main.c file):



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is `avas605@vx3:~/maths_module$`. The first command is `dcc maths.c main.c -o maths`. The second command is `./maths`. The output shows two lines of text: "The square of the number 13 is 169" and "The sum of 13 and 10 is 23". The prompt returns to `avas605@vx3:~/maths_module$`. Two red arrows from the text above point to `maths.c` and `main.c` in the command line.

```
avas605@vx3:~/maths_module$ dcc maths.c main.c -o maths
avas605@vx3:~/maths_module$ ./maths
The square of the number 13 is 169
The sum of 13 and 10 is 23
avas605@vx3:~/maths_module$
```

- The program will always enter in main.c, so there should only be one main.c when compiling



Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

<https://www.menti.com/na5yrb8evn>

WHAT DID WE LEARN TODAY?

LINKED LISTS

linked_list.c

main.c

linked_list.h

ice_cream_shop.c

MULTI-FILE PROJECTS

maths.c

main.c

maths.h

REACH OUT



CONTENT RELATED QUESTIONS

Check out the forum



ADMIN QUESTIONS

cs1511@cse.unsw.edu.au