COMP1511 PROGRAMMING FUNDAMENTALS

Lecture 3

Getting harder...

More complex IF statements, a closer look at scanf(), breaking things, and learning about STRUCTS

LAST WEEK...

- Started looking at C
- Our first Hello! program
- Compiling and running your code
- printf() and scanf()
- Variables (int, char, double)
- Maths :)
- Basic IF statements

TODAY...

- More complex IF statements
- Logical operators
- Chaining IF and ELSE
- Breaking stuff

- Structs

WHERE IS THE CODE?

LIVE LECTURE CODE CAN BE FOUND HERE:

https://cgi.cse.unsw.edu.au/~cs1511/21T3/live/Week02/

IF / ELSE IF / ELSE

LET'S LOOK AT SOME CODE AND A DEMO

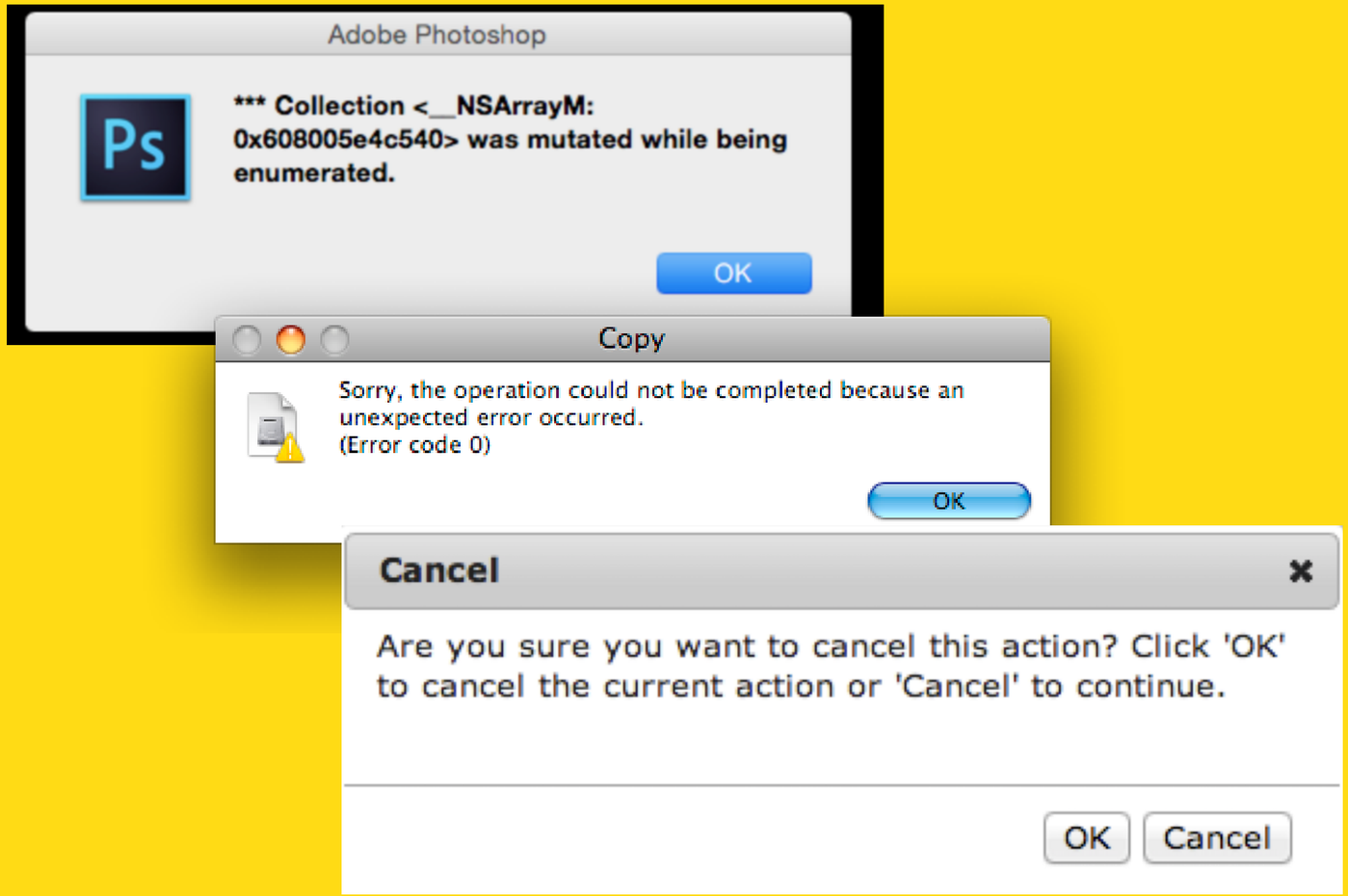
- IF statements with logical operators:
  if_logic.c
- IF statements with char:
  upper.c
- Harder IF logic and chaining if and else together:
  dice_checker.c

BREAKING THINGS

It is really good practice to think about how it is possible to break your code? What can go wrong?

BREAKING THINGS

IS ALWAYS FUN



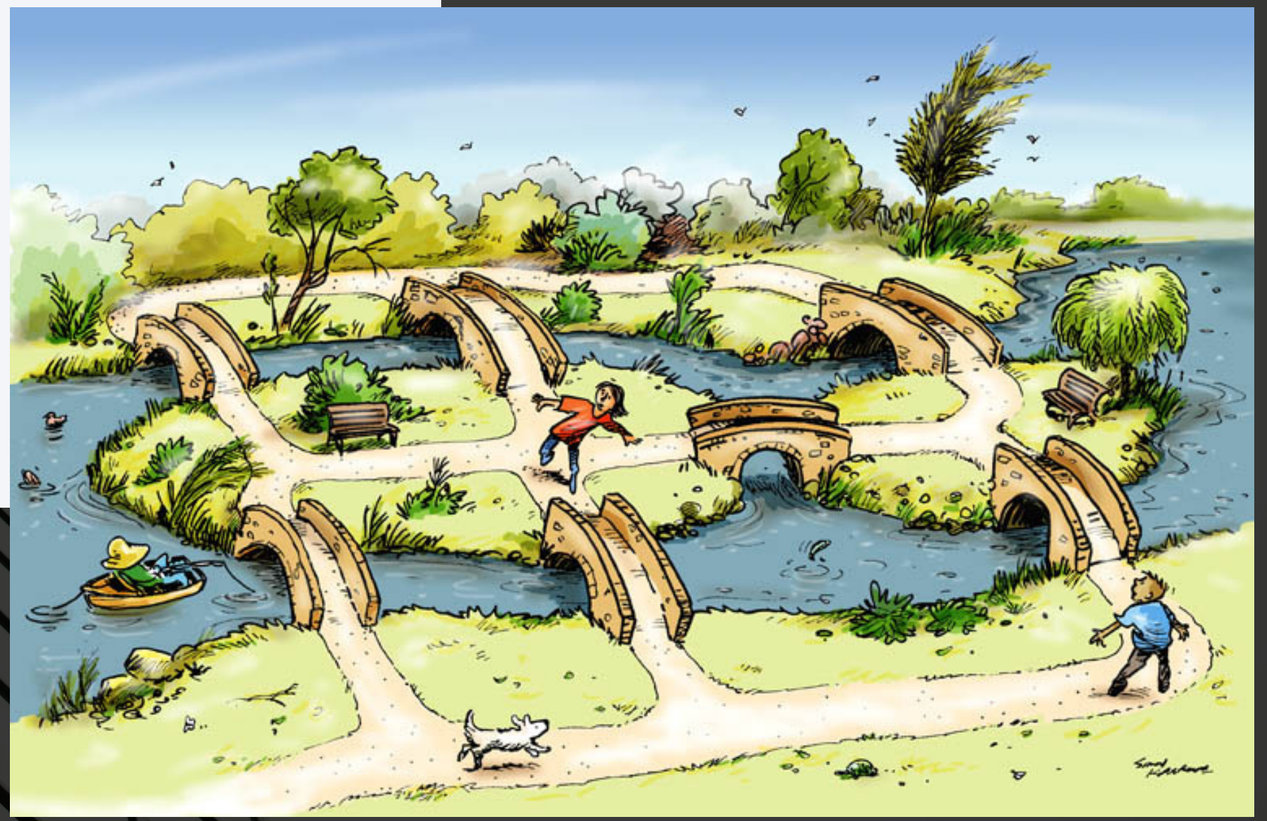
- Try and counter for these breaks!
- Important to have good error messages:
  - Tells the user exactly what has gone wrong
  - How can they fix it?
  - What is happening!?

BREAK TIME (5 MINUTES)

Have you ever heard of the Bridges of Konigsberg?

Below is an image of the seven bridges and their positions (Image credit).

Is it possible to walk around crossing each bridge only once?



HOW DOES SCANF() REALLY WORK?

A MAGICAL POWER...

- Gives us the ability to scan stuff in from the terminal (standard input)
- We have to tell the computer what we expect to scanf() - is it an *int*, a *double*, or a *char*?
- But since *scanf()* is a function does it return something?
  - Yes, *scanf()* returns the number of input values that are scanned
  - If there is some input failure or error then it returns EOF (end-of-file)
  - This is useful to check for any errors

DID YOU NOTICE HOW A NEW LINE IS READ BY SCANF()?

BECAUSE /N IS A CHARACTER ON THE ASCII TABLE: 10 LF (LINE FEED)

- You may have noticed that scanf("%d", &number) is able to ignore anything other than a number when it scans in - this is because whitespace is not a number and the function looks for a number
- But did you notice that this is not the case for scanf("%c, &character)? This is because a new line (/n) is a character on the ASCII table, which means it is still a valid character to scan in (It is number 10 LF if you are interested!)
- To fix this, we can tell scanf() to ignore all preceeding whitespace by using a special magic trick:
  scanf(" %c", &character)

ORGANISING DIFFERENT TYPES INTO ONE RELATED WHOLE

USER DEFINED DATA TYPE: STRUCT

- Structures…. Or **struct** (as they are known in C!)
- Structs (short for structures) are a way to create custom variables
- Structs are variables that are made up of other variables

STRUCTURES

WHAT? WHY? EXAMPLES?

- What happens if you wanted to group some variables together to make a single structure?
- Why do we need structures?
  - Helps us to organise related but different components into one structure
  - Useful in defining real life problems
- What are some examples in real life where some things go together to make a single component?

HOW DO WE CREATE A STRUCT?

To create a struct, there are three steps:

1. Define the struct (outside the main)
2. Declare the struct (inside your main)
3. Initialise the struct

1.DEFINING A STRUCT

WHAT AM I GROUPING TOGETHER INTO ONE WHOLE? LET'S USE AN EXAMPLE OF A COORDINATE POINT

Because structures are a variable that we have created, made up of components that we decided belong together, we need to define what the struct (or structure is). To define a struct, we define it before our main function and use some special syntax.

```
struct struct_name {

data_type variable_name_member;
data_type variable_name_member;
...

};
```

```
struct coordinate {

int x_coord;
int y_coord;

};
```

2.DECLARING A STRUCT

INSIDE YOUR MAIN

- To declare a struct, inside the main function (or wherever you are using the structure - more on this later)...

```
struct struct_name variable_name;
```

```
struct coordinate cood_point;
```

3.INITIALISE A STRUCT

INSIDE YOUR MAIN

- We access a member by using the dot operator .

```
variable_name.variable_name_member;  ⟶  cood_point.x_coord
```

LET'S SEE IT ALL TOGETHER FOR A COORDINATE POINT

1. DEFINE
2. DECLARE
3. INITIALISE

1.define

[outside the main]

```
struct coordinate {

    int x_coord;
    int y_coord;

};
```

2.declare

[inside the main]

```
// Declare structure with variable name

struct coordinate cood_point;
```

3.initialise

[inside the main]

```
// Access struct member to assign value

cood_point.x_coord = 3

cood_point.y_coord = 6
```

LET'S SEE STRUCTS IN ACTION

CODE DEMO

- You can see structs in action (I feel like we are in some sort of epic film here):

  structs_intro.c

WHAT DID WE LEARN TODAY?

LOGICAL OPERATORS AND IF WITH CHAR

upper.c

CHAINING IF/ELESE AND ERROR CHECKING

dice_checker

SAY HELLO TO STRUCTS

structs_intro.c

ANY QUESTIONS?

DON'T FORGET YOU CAN ALWAYS EMAIL US ON CS1511@CSE.UNSW.EDU.AU FOR ANY ADMIN QUESTIONS

PLEASE ASK IN THE FORUM FOR CONTENT RELATED QUESTIONS