LECTURE 2

Throwing ourselves into the thick of it: Variables and IF Statements
ON TUESDAY, WE TALKED:

- Welcome and Introductions
- Course Administration
- How COMP1511 works
- How to get help and the best ways to approach learning Programming
- What is programming?
- What is Linux and working in Linux
IN THIS LECTURE

TODAY...

• Variables and how we store information
• Maths in C!
• Organising code
• Conditionals - running out code based on some sort of condition being met
• IF statements
WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/22T2/LIVE/WEEK_01/
A BRIEF RECAP

OUR FIRST PROGRAM

```c
#include <stdio.h>

int main (void) {
    printf("Hey!\n");
    return 0;
}
```
ORGANISING CODE

PROCEDURES

- Help us group functionality into sections of code.
- Allow us to re-use code without re-typing it.
- You’ve already used one (sort of)

```c
void my_procedure(void) {
  // this is a procedure, called my_procedure.
}
```
Computer memory is literally a big pile of on-off switches
  - We call these bits (smallest possible unit in computing, a bit is a choice between two things a 0 or a 1)

We often collect these together into bunches of 8 bits
  - We call these bytes
When we execute code, the CPU will actually process the instructions and perform basic arithmetic, but the RAM will keep track of all the data needed in those instructions and operations.
**WHAT IS A VARIABLE?**

- Our way of asking the computer to remember something for us
- Called a "variable" because it can change its value
- A certain number of bits that we use to represent something
- Made with a specific purpose in mind
We're going to start out with three data types of variables:

- **int**: integer, a whole number (eg: 0,1,2,3)
- **char**: a single character (eg. ‘a’, ‘A’, etc)
- **double**: floating point number (eg: 3.14159, 8.534, 7.11)

Each of these has a different number of bytes that are allocated in memory once the program is run...
• Names are a quick description of what the variable is
  ○ Eg: “answer” and “diameter”
  ○ Rather than “a” and “b”
• We always use lower case letters to start our variable names
• C is case sensitive:
  ○ “ansWer” and “answer” are two different variables
• C also reserves some words
  ○ “return”, “int” and “double” can’t be used as variable names
• Multiple words
  ○ We can split words with underscores: ”long_answer”
We name our variables in ways that make it obvious what they are representing. Remember someone else has to be able to skim your code and know what you are saying/doing!
INTEGER

DATA TYPE int

- A whole number, with no fractions or decimals
- Most commonly uses 32 bits (which is also 4 bytes)
- This gives us exactly $2^{32}$ different possible values
- The maximum is very large, but it’s not infinite!

Exact ranges from -2147483648 ($-2^{31}$) to 2147483647 ($2^{31} - 1$)
A single character in C can also be represented as an int!

This is because a single character variable holds an ASCII value (integers 0-127), as opposed to the character itself.

The syntax to assign a single character is to put the character in single quotes: ‘a’.

So for a capital letter A:, the character is ‘A’ and the int stored is 65.

You use a char to declare a character: char letter = ‘a’ - this will assign 97 to the variable letter.
• A double-sized floating point number
• A decimal value - "floating point" means the point can be anywhere in the number
• Eg: 10.567 or 105.67 (the points are in different places in the same digits)
• It’s called "double" because it’s usually 64 bits, hence the double size of our integers (or 8 bytes)
Let's try some code. Declare and initialise a variable.

```c
int main (void) {
    // Declaring a variable
    int answer;
    // Initialising the variable
    answer = 42;
    // Give the variable a different value
    answer = 7;
    // We can also declare and initialise together
    int answer_two = 88;
}
```
Not just for specific messages we type in advance

We can also print variables to our display!

To print out a variable value, we use format specifiers

- this is a % symbol followed by some characters to let the compiler know what data type you want to print..
- %d where the output you'd like to put an int (decimal value, hence %d)

After the comma, you put the name of the variable you want to write

```c
// Printing a variable
int number = 13;
printf("My number is \%d\n", number);
```
The variables will match the symbols in the same order as they appear!

You can have as many as you want and of different types also!

```
21 // Printing out two variables
22 int first = 3;
23 int second = 10;
24 printf("First is %d and second is %d\n", first, second);
```
**LET'S TRY DIFFERENT TYPES OF NUMBERS**

INTS AND DOUBLES - OH MY!

- The `%d` and `%lf` are format specifiers that are used in printf statement to let the compiler know what data type we need to output.
  - `%d` stands for “decimal integer”
  - `%lf` stands for “long floating point number” (a double)

- Remember that we have to be very prescriptive when we tell the computer what to do and that extends to even telling it what types we are printing in C

```c
// print an int and a double
int diameter = 5;
double pi = 3.141;
printf("The diameter is %d, pi is %lf\n", diameter, pi);
```
The `%c` format specifier can also be used in printf statement to let the compiler know what data type we need to output (character).

- `%c` stands for “character”
- Don’t forget that when you declare a char, you enclose it in single apostrophes to let the computer know that you are using a letter character

```c
// print an int as a character
char letter = 'A';
printf("The letter %c has the ASCII value %d\n", letter, letter);
```
GREAT, WE CAN PRINT TO TERMINAL, CAN WE READ SOMETHING FROM TERMINAL?

- Reads input from the user in the same format as printf
- Format specifiers (\%d, \%lf, \%c) are used in the same way as for the printf statement
- The & symbol tells scanf the address of the variable in memory (where the variable is located) that we want to place the value into (more details later in term)

```c
// reading an integer
int input;
printf("Please type in a number: ");
scanf("\%d", &input);

// reading a double
double decimal;
printf("Please type in a decimal number: ");
scanf("\%lf", &decimal);
```
If you want scanf to read in a character, you will need to declare a character by using the keyword: `char`

Even though you have declared a char to store your character into, it is still stored as an ASCII value... so you can move between `%d` and `%c` when you printf this variable

```c
10 // reading a single character as a character
11 char character;
12 printf("Please type in a character: ");
13 scanf("%c", &character);
```
 Constants are like variables, only they never change!

To define a constant, we use `#define` and follow it with the name of the constant and the value.

```
#define PI 3.1415
#define MEANING_OF_LIFE 42
#define MAX_NUMBER 13

int main (void) {
```

Style Guide: We name them in all caps so that we remember that they're not variable.
A lot of arithmetic operations will look very familiar in C
- adding +
- subtracting -
- multiplying *
- dividing /

These will happen in their normal mathematical order

We can also use brackets to force precedence

```c
16 int x = 5;
17 int y = 10;
18 int result;
19 result = (x + y) * x;
20 printf("Result is %d\n", result);
```
Because characters are represented as ints inside the variable, you are able to move around the ASCII values by adding or subtracting to them.

For example, if you are at ‘a’ and you want to get to ‘b’, you can add 1

```
// some basic Maths
char letter = 'a';
char next_letter = letter + 1;
printf("My original letter %c with ASCII value %d\n", letter, letter);
printf("The next letter %c with ASCII value %d\n", next_letter, 1
next_letter);
```
Check out Boeing 787 that had to be rebooted every 248 days (2^31 hundredths of a seconds) https://www.engadget.com/2015-05-01-boeing-787-dreamliner-software-bug.html

If we add two large ints together, we might go over the maximum value, which will actually roll around to the minimum value and possibly end up negative (Check out Ariane 5 explosion), a simple error like this caused a rather large problem: https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster)

THE QUIRKS OF INTEGERS...

INTEGER OVERFLOW/ INTEGER UNDERFLOW
In a less destructive example, the video Gangham Style on YouTube maxed out the views counter:

ints might not always be 32 bits . . . dependent on Operating System
THE QUIRKS OF DOUBLES...

- No such thing as infinite precision
- We can't precisely encode a simple number like $\frac{1}{3}$
- If we divide 1.0 by 3.0, we'll get an approximation of $\frac{1}{3}$
- The effect of approximation can compound the more you use them
Remember that C thinks in data types
  ○ If either numbers in the division are doubles, the result will be a double
  ○ If both numbers are ints, the result will be an int, for example, 3/2 will not return 1.5, because ints are only whole numbers
  ○ ints will always drop whatever fraction exists, they won't round nicely, so 5/3 will result in 1
• % is called Modulus. It will give us the remainder from a division between integers, eg. 5 % 3 = 2 (because 5/3 = 1 rem 2)
There has just been a heavy fall of snow, Baudouim goes outside and finds that there is twice as much snow in his garden as in his neighbour Gael’s garden. He does not, however, appear surprised. Why not?
Sometimes we want to make decisions based on what information we have at the time. We can let our program branch between sets of instructions. In C this is the `if` statement.
A decision problem is a question with a YES/NO answer.

This is the perfect time to use an IF statement to help make the decision.

Eg. Is a number even? Is a number larger than 10? Is a number prime? etc.
First we ask the question - this is our condition
If the answer to our question (condition) is YES, then we run the code in the curly brackets

```java
// the code inside the curly brackets
// runs if the expression is true (not zero)
if (condition) {
    code statement;
    code statement;
}
```
If the answer to our question (condition) is NO, then we can add an else statement to let the computer know which other code may run.

```java
if (condition) {
    // code to run if the condition is true
    // or anything other than 0
} else {
    // run some other code instead
    // else is entered if the previous code
    // results in 0 (false)
}
```
If the answer to our question (condition) is NO, and the answer to our question (condition) in the else is also NO, then we can chain some if and else together to make an else if and create even more options in choosing which code to run...

```java
if (condition1) {
    // code to run if condition1 is true
    // (anything other than 0)
} else if (condition2) {
    // code to run if condition1 is false (results in 0)
    // and condition2 is true (results in anything other than 0)
} else {
    // code to run if both condition1 and condition2 result in false (0)
}
```
Relational Operators work with pairs of numbers:
  - < less than
  - > greater than
  - \(\leq\) less than or equal to
  - \(\geq\) greater than or equal to
  - == equals
  - != not equal to

All of these will result in 0 if false and a 1 if true
SOME EXAMPLES

LET'S TRY THIS OUT...

• True (1) or False (0)?

```cpp
if (12 <= 12) {
    //do something
}
```

```cpp
if (8 != 8) {
    //do something
}
```

```cpp
if (5 < 10) {
    //do something
}
```
I LIKE QUESTIONS, HOW DO I ASK TWO QUESTIONS AT THE SAME TIME?

LOGICAL OPERATORS

The first two are used between two questions (expressions):

- **&&** AND: if both expressions are true then the condition is TRUE (equates to 1 if both sides equate to 1)
- **||** OR: if any of the two expressions are true then the condition is TRUE (is 1 if either side is 1)

This is used in front of an expression:

- **!** NOT: reverse the expression (is the opposite of whatever the expression was)
**SOME EXAMPLES**

**LET'S TRY THIS OUT...**

- True (1) or False (0)?

```java
if (7 < 15 && 8 >= 15) {
    //do something
}
```

```java
if (7 < 15 || 8 >= 15) {
    //do something
}
```

```java
if !(5 < 10 || 6 > 13) {
    //do something
}
```
A user rolls two dice and tells us the number on each of the rolled die. Our program will add the die numbers together and check them against a target number that only the program knows. It will then report back whether the total of the dice was higher, equal or lower than the secret number.

LET'S PUT OUR SKILLS TO THE TEST

LET'S CODE! (SOLVE THE PROBLEM FIRST)
A user will roll two dice - done outside of our program

1. Take in the result of each die - how do we read input?
2. Add the die numbers together
3. Check them against a target number - based on steps 4 and 5, it looks like we need to make a decision - therefore IF statement
4. Output if total of the dice was higher, equal or lower than the target number - output based on the decision that we made
BREAKING DOWN THE PROBLEM INTO A SUM OF SIMPLE PARTS

A user rolls two dice and tells us the number on each of the rolled die. Our program will add the die numbers together and check them against a target number that only the program knows. It will then report back whether the total of the dice was higher, equal or lower than the secret number.

1. Take in the result of each die - how do we read input?
   a. Read input of die 1
   b. Read input of die 2
2. Add the die numbers together
   ○ sum = die1+die2
3. Check them against a target number - based on steps 3 and 4, it looks like we need to make a decision - therefore IF statement
   ○ Define the target number
4. Output if total of the dice was higher, equal or lower than the target number. - output based on the decision that we made
   ○ Is sum greater than target number?
   ○ Is sum less than target number?
   ○ Is sum equal to the target number?
NOW LET'S CODE!

1. Switch over to VLab
2. Open Terminal
3. Open a new file:
   
   ```
   gedit dice_checker.c &
   ```

Feel free to follow along with lecture coding, or you can also find the code here:
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://forms.microsoft.com/r/dKssTn3AU4
WHAT DID WE LEARN TODAY?

RECAP
- Hello World!
- Our first program

VARIABLES
- They come in different shapes and sizes - int, double and char
- Printing from variables (printf)
- Reading user input into variables (scanf)
- Using maths with variables

CONDITIONS
- if / else / else if
- Decision problems
- Relational Operators
- Logical Operators

DICE_CHECKER
- Putting it all together in code
REACH OUT

CONTENT RELATED QUESTIONS
Check out the forum

ADMIN QUESTIONS
cs1511@cse.unsw.edu.au