COMP1511/1911 Programming Fundamentals

Week 2 Lecture 1

Control Flow

Last Week

- You went to your first tut/lab
- compiling and running hello_world.c
- variables and types (`int`, `double`, `char`)
- `printf()` and `scanf()`
- Arithmetic operators and expressions (maths calculations)
- We did not finish Constants from last weeks Thursday Lectures

Labs this week

- Week 1 lab was not worth any marks
- Week 2 lab is!
  - Exercises are due to be submitted by Monday 8pm the following week
  - 3 dot questions are challenge questions and are sometimes very difficult and time consuming. These are not essential to get full marks in the labs.

Today's Lecture

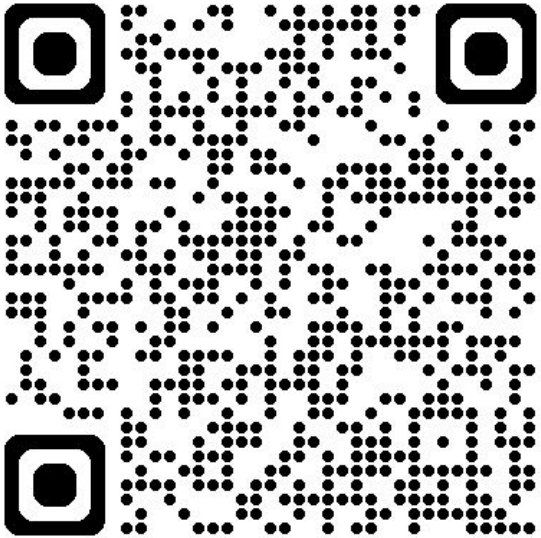
Recap of scanf, printf and expressions from last thursday.
Constants, that we did not get to last thursday

Getting harder...

- `if` statements
- relational and logical operators
- error checking scanf input
- `while` loops (if there is time...)

Link to Week 2 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/24T3/live/week_2/



A Brief Recap: Variables and printf

```c
int number_of_weeks = 10;
double distance_in_cm = 95.5;
char grade = 'B';

printf("There are %d weeks\n", number_of_weeks);
printf("The distance is %lf cm\n", distance_in_cm);
printf("You got a %c!\n", grade);

distance_in_cm = distance_in_cm - 5;
printf("The distance is now %lf cm\n", distance_in_cm);
```

A Brief Recap: Variables scanf

```c
int x;
int y;
printf("Enter 2 integers: ");
scanf("%d %d", &x, &y);
printf("You entered %d %d\n", x, y);

double real;
printf("Enter a real number: ");
scanf("%lf", &real);
printf("You entered %lf\n", real);
```

A Brief Recap: Variables scanf

- Can you spot the difference between the 2 code fragments below?
- How would the code fragments behave differently?

```c
char character;
printf("Enter a character: ");
scanf("%c", &character);
printf("%c\n", character);
```

```c
char character;
printf("Enter a character: ");
scanf(" %c", &character);
printf("%c\n", character);
```

A Brief Recap: What will this do?

```c
int x;
int y;
printf("Enter 2 integers: ");
scanf("%d %d", &x, &y);
int average = (x + y) / 2;
printf("The average of %d and %d is %d\n", x, y, average);
```

```c
int z = 5 / 0;
char letter_1 = 'B' - 'A' + 'a';
char letter_2 = 'e' - 'a' + 'A';
```

Constants

- Constants are like variables, only they never change!
- We use **`#define`** and follow it with the name of the constant and its value
- Style Guide: we use UPPERCASE so it is easy to recognise they are constants

```c
#include <stdio.h>
// Define constants after your #includes
// but before your main
#define MAX_SIZE 12
#define PI 3.1415
#define MEANING_OF_LIFE 42
```

Coding Time

Write a program **`convert.c`** that

- prompts the user enter the number of hours
- calculates how many minutes that is equivalent to
- prints out the number of minutes

See sample output below:

```
$ ./convert
Please enter the number of hours: 2.5
That is 150.00 minutes
```

Making Decisions with if

- `if` statements allow C to make decisions based upon true/false questions like
  - is x even
  - is y greater than 10
  - is x less than y
- true and false are ints in C
  - 0 is false
  - 1 or any non-zero value is considered true in C

if statement

- Our true/false question is called a condition
- If the answer to our question i.e. our condition is true then we run the code inside the `if` block's braces **{}**

```
// The code inside the curly brackets only runs if
// the condition is true
if (condition) {
    // do something;
    // do more things;
}
```

Relational Operators

- To write conditions we need to be able to compare things
- Relational Operators compare pairs of numbers
- All of these will result in 0 if false and a 1 if true
- Be careful not to mix up = and ==
  - Use = for assigning a value
  - Use == for comparing 2 values
- Be careful using == and != for double values

Operator	
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
==	Is equal to
!=	Not equal to

Quick Questions: Relational Operators

true (1) or false (0):

```
int x = 4;
```

- 5 < 2
- x > 2
- x <= 4
- 3 >= x
- x == -4
- 'A' != 'B'

What will get printed?

```c
if (1) {
    printf("Hooray\n");
}

if (0) {
    printf("Yay!\n");
}

if (4 == 4) {
    printf("I love C!\n");
}

if ('Z' == 'z') {
    printf("I am cool!\n");
}
```

```c
int x = 5;
int y = 10;

if (x < 0) {
    printf("x is negative!\n");
}

if (y >= x) {
    printf("y is greater or equal\n");
}

if (x != y) {
    printf("x and y not equal!\n");
}
```

if-else statement

- We can add an **`else`** statement to run a block of code when our condition is not true.

```
if (condition) {
    // this code runs if condition
    // is true  (non-zero)
} else {
    // this code runs if condition
    // is false (0)
}
```

if-else statement example

```
#define COLD 10
```

```
if (temperature <= COLD) {
    printf("I am cold!\n");
} else {
    printf("I am not cold!\n");
}
```

Chaining if statements

We can chain multiple if statements to check for multiple options

```
if (condition_1) {
    // this code runs if condition_1 is true (non-zero)
} else if (condition_2) {
    // this code runs if condition_1 was false
    // and condition_2 is true (non-zero)
} else {
    // this code runs if condition_1 and condition_2
    //  are both false (0)
}
```

Chaining if-statements example

```
#define COLD 10
#define HOT 25
```

```
if (temperature <= COLD) {
    printf("I am cold!\n");
} else if (temperature < HOT) {
    printf("Just right!\n");
} else {
    printf("I am hot!\n");
}
```

Coding Time

Write a program **`odd_even.c`** that

- prompts the user enter an integer
- prints out whether the number is even or odd

```
$ ./odd_even
Please enter an integer: 8
Even!
./odd_even
Please enter an integer: 3
Odd!
```

Coding Time

Write a program **`guessing_game.c`** that

- prompts the user enter an integer
- prints out higher, lower, correct if it is equal to the secret number (42)

```
$ ./guessing_game
Please enter an integer: 8
Higher!
$ ./guessing_game
Please enter an integer: 42
Correct!
```

Nested If Statements

- We can have nested if statements
  - These are if statements inside if statements!
- We don't want too much nesting though for style!
  - Maximum 5 levels
- Note the indentation
  - Every time we have **{** we indent a level
  - Every time we have **}** we indent one less level

```
if (x > 0) {
    if (x > 100) {
        printf("big");
    } else {
        printf("small");
    }
    printf(" positive");
}
```

Quick Break

Logical Operators

Often we want to ask more than one true or false question at the same time

- E.g. Is x greater than y and less than z?

Operator	Name	Explanation	Example usage
`&&`	and	true if both operands are true	`x > y && x < z`
`||`	or	true if least one operand is true	`x == MAX || y == MAX`
`!`	not	true if the operand is false	`! (x >= 0)`

Truth Tables

- Truth tables show the results of logical operators with all different combinations of possibilities

C1	C2	C1 && C2
0	0	0
0	1	0
1	0	0
1	1	1

C1	C2	C1 || C2
0	0	0
0	1	1
1	0	1
1	1	1

C	!C
0	1
1	0

Logical Operators

Warning: Don't do something like 0 < x < 10

- It might compile and run but probably does not do what you want
- It is no checking whether x is between 0 and 10
- Do something like this instead: `x > 0 && x < 10`

Warning: Beginners sometimes get confused with && and ||

- `((x > 0) || (x < 10))`
  - This is always true
- `((x < 0) && (x > 10))`
  - This is always false

Quick Questions: Logical Operators

true (1) or false (0)

```
char c = 'g';
int x = 4;
```

- `(c >= 'a') && (c <= 'm')`
- `  (x < 0)  || (x > 10)`
- `!((x < 0)  || (x > 10))`

Short circuit Evaluation

Do we need to know the value of y to know if these conditions are true?

```
int x = 3;
int y = ?;
if ((x == 3) || (y == 5)) {
```

```
int x = 4;
int y = ?;
if ((x == 3) && (y == 5)) {
```

Short Circuit

- The operators `&&` and `||` evaluate
  - their left-hand-side operand first and only evaluate their
  - right-hand-side operand if necessary
- Operator `&&` only evaluates its RHS if the LHS is true
- Operator || only evaluates its RHS if the LHS is false
- This is very useful because we can safely write:

```
if ((x != 0) && ((y / x) > 10))
```

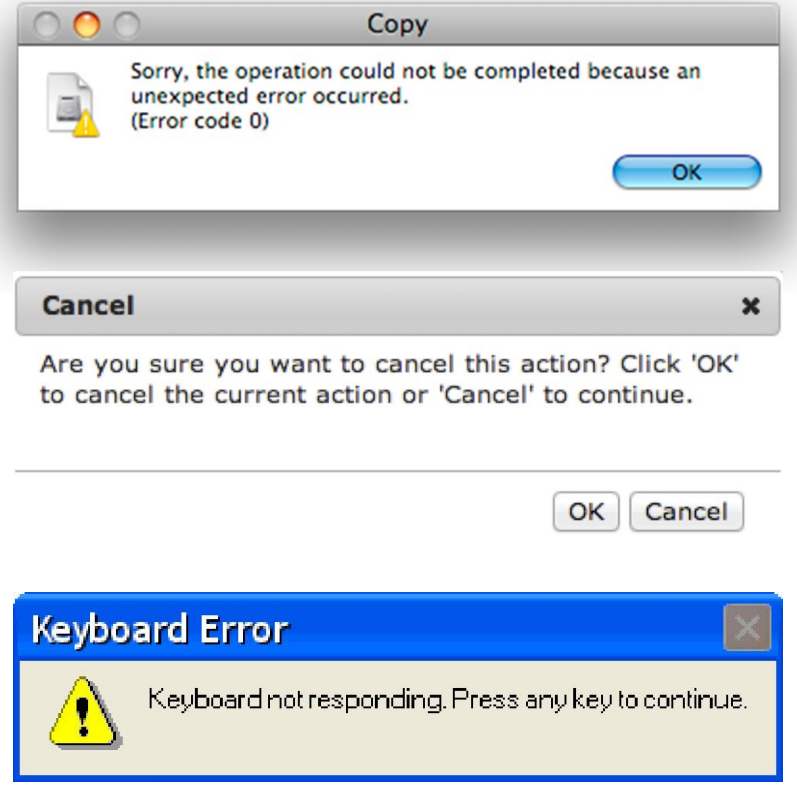
Coding Time with Logical Operators

`character_cases.c`

Write a program to allow a user to enter a character
Print out whether the character is an uppercase letter,
a lowercase letter, or not a letter.

What test cases should we make sure we check?

Breaking Things



It is really good practice to think about how it is possible to break your code?

- What can go wrong?
- Important to have good error messages:
  - Tells the user exactly what has gone wrong
  - How can they fix it?
  - What is happening!?

Can we check for input errors from scanf?

scanf return value and error checking

- We tell scanf what we want it to scan in
  - But what if the user types in the wrong type of data?
- scanf has a way of telling us!
  - It returns the number of inputs it scanned in successfully and we can assign it to a variable and/or use it for error checking.

```
int inputs_read = scanf("%d %d", &x, &y);
if (inputs_read != 2) {
    printf("Incorrect input\n");
}
```

Coding Time

- Let’s modify guessing_game.c to check that user entered a valid integer…

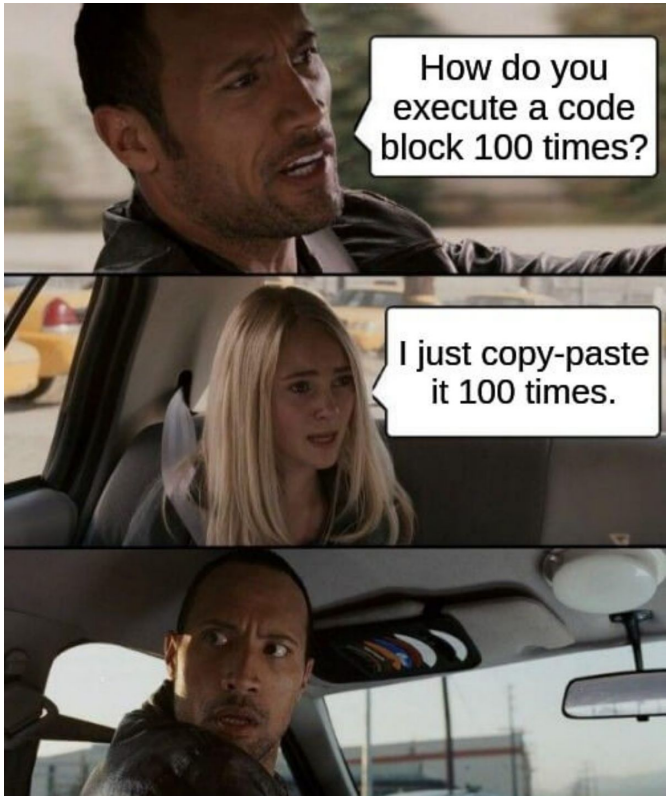
while bowl of chips is not empty
    each chip

While Loops

while in lecture
    stay awake

Repetition, Repetition, Repetition in C

- C normally executes line by line
- if statements allow us to select parts of our code
- But how can we repeat code?
- Copy-pasting the same code again and again is not a feasible solution



While Loops

- While something is true, do something
- Here is the general while loop syntax

```
while (condition) {
    // Code in here runs again and again
    // until the condition is false
    // The program will jump back to the start
    // of the while loop when it gets to the closing
}
```

Infinite while Loops

- It is very easy to make an infinite while loop
  - Handy Tip: Type Ctrl + C to end infinite while loops

```c
while (1) {
    printf("I love my COMP1511 lectures!\n");
}
```

```c
int push_ups = 0;
while (push_ups < 100) {
    printf("You have done %d push-ups!\n", push_ups);
}
```

3 Ways of Controlling while loops

- counting loops
  - The number of iterations is known
  - Use a variable as a counter to control how many times a loop runs
- conditional loops
  - We may not know how many times we will need to loop
  - Conditions terminate the loop based on calculations or user input
- sentinel loops
  - Special case of conditional loops
  - A sentinel loop continues to execute until a special value (the sentinel value) is encountered.

Counting while loops

- Use a loop control variable (“loop counter”) to count loop repetitions.
  - We stop when the loop reaches a certain limit.
- Useful when we know how many iterations we want.

```
// 1. Initialise loop counter before the loop
int counter = 0;
while (counter < 10) { // 2. check loop counter condition
   printf("Here we go loop de loop!\n");
   counter = counter + 1; // 3. update loop counter
}
```

Conditional Loops

- Iterate as long as your condition is still true
- Used when we don't know how many times we need to loop

```c
// 1. Initialise the loop control variable
int total_kombucha_ml = 0;
int kombucha_ml;
while (total_kombucha_ml <= 2000) { // 2. Test the loop condition
    printf("Please enter the ml of kombucha: ");
    scanf("%d", &kombucha_ml);
    // 3. Update loop control variable
    total_kombucha_ml = total_kombucha_ml + kombucha_ml;
}
printf("Stop! That would bring you to %dml!!\n", total_kombucha_ml);
```

Sentinel Loops

- Process data until reaching a special value (sentinel value)
  - Special case of conditional loop

```c
int number = 0;
int end_loop = 0;        // 1. Initialise the loop control variable
while (end_loop == 0) { // 2. Test the loop condition
    scanf("%d", &number);
    if (number < 0) {    // We want a negative value to end the loop
        end_loop = 1;    // 3. Update the loop control variable
    } else {
        printf("You entered %d\n", number);
    }
}
```

Code Demo

`while_infinite.c`
`while_count.c`
`while_condition.c`
`while_sentinel.c`

Write a program that reads integers from the user and sums them until a non-integer input is encountered

`while_scanf_sum.c`

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/vKrxXCYzPR

What did we learn today?

- Recap and Constants (convert.c)
- If statements
  - Conditions and relational operators (odd_even.c guessing_game.c)
  - Conditions and Logical Operators (character_cases.c)
  - Error checking scanf input (guessing_game.c)
- While Loops
  - while_infinite.c, while_count.c, while_conditional.c, while_sentinel.c, while_scanf_sum.c

Reach Out

Content Related Questions:
Forum

Admin related Questions email:
cs1511@unsw.edu.au