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)

Labs this week

- Week 1 lab was not worth any marks
- Week 2 lab is!
  - Exercises are due Monday 6pm 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

Quick recap

- scanf, printf and expressions
- #define constants we did not cover last week

Make decisions in our code

- **if** statements
  - relational and logical operators
  - checking scanf input

Use the power of repetition.

A small amount of code can do a lot of work!

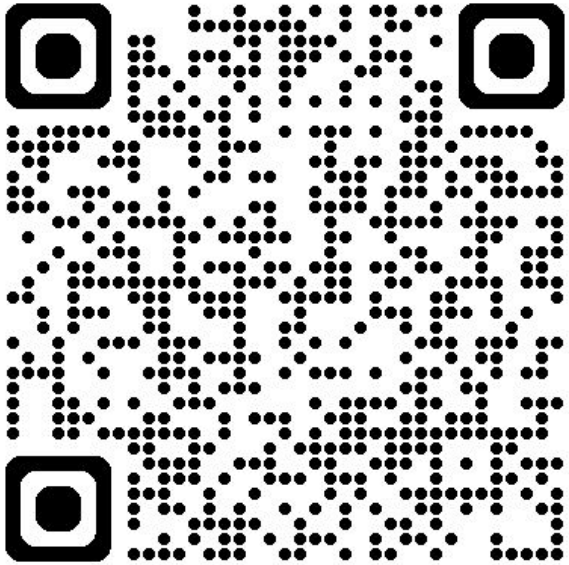
- **while** loops

Things are Getting Harder...

Keep practising!

Link to Week 2 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/26T3/code/week_2/



Recap: Printing out variables with printf

- A format specifier starts with **%**
- It tells **printf** what type of value to print
- Put the variables after the format string

Type	Format specifier
`int`	`%d`
`double`	`%lf`
`char`	`%c`

```c
int age = 21;

printf("My age is %d\n", age);
```

A Brief Recap: Variables and printf

```c
int gold = 10;
double health = 85;
char initial = 'X';

printf("Player: %c\n", initial);
printf("Gold: %d Health: %lf\n", gold, health);

gold = gold + 1;
health = health / 2;
printf("Gold: %d Health: %lf\n", gold, health);
```

A Brief Recap: Variables scanf

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

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

A Brief Recap: Some Tricky Expressions

What values will be stored in each variable?

```c
int x = 11;
int remainder = x % 3;
int y = x / 2;
char letter = 'B' - 'A' + 'a';
```

Constants

- A constant is a name for a value that should not change
  - Put **#define** after the includes and before **main**
  - Use UPPERCASE names so constants are easy to recognise
  - Less error prone. Define the value once, use the name everywhere

```c
#include <stdio.h>

#define MAX_SIZE 12
#define PI 3.1415
#define MEANING_OF_LIFE 42
```

Magic Numbers Example

- What is going on?
- What do these numbers mean?
- These are called **magic numbers**.
- Constants can give these numbers meaningful names

```c
int gold = 100;
gold = gold - 40;
gold = gold + 50;
gold = gold - 30;
printf("Gold: %d\n", gold);
```

Magic Numbers Example

We can use #define constants to replace magic numbers.

```c
#define STARTING_GOLD 100
#define SWORD_COST 40
#define SHIELD_COST 30
#define POT_OF_GOLD 50
```

```c
int gold = 100;
gold = gold - 40;
gold = gold + 50;
gold = gold - 30;
printf("Gold: %d\n", gold);
```

```c
int gold = STARTING_GOLD;
gold = gold - SWORD_COST;
gold = gold + POT_OF_GOLD;
gold = gold - SHIELD_COST;
printf("Gold: %d\n", gold);
```

Control Flow

Making Decisions with if

- `if` statements allow C to make decisions based on **true/false** questions like
  - is x even
  - is y greater than 10
  - is x less than y
- In C
  - 0 means false
  - 1 or any non-zero value means true

Relational Operators

- Relational Operators compare two values
- They evaluate to
  - `0` if false
  - `1` if true
- Be careful not to mix up `=` and `==`
  - Use `=` to assign a value
  - Use `==` to compare 2 values
- Be careful using `==` and `!=` with 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

What do these evaluate to: true (1) or false (0)?

```
int x = 4;
```

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

if statement

- Our true/false question is called a condition
- If the condition is **true**, the code inside the braces {} runs

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

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");
}
```

What will get printed?

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

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

if (y >= x) {
    printf("y is greater than or equal to x\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 the 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

```c
#define COLD 10
```

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

Chaining else if statements

We can chain multiple **`else if`** statements to check multiple conditions in order

```
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 else 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 Example: odd_even.c

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

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

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

Coding Example: guessing_game.c

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

- Prompts the user to enter an integer
- Prints **Higher!**, **Lower!**, or **Correct!** depending on the guess
- The correct answer is always 42.

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

Quick Break. Back Soon…

Nested If Statements

- We can have **nested** if statements
  - These are if statements inside if statements!
  - The inner only runs if the outer condition is true
- Avoid too much nesting!
  - **Maximum 5** levels for style
- Note the **indentation**:
  - **{** indent one extra level
  - **}** indent one less level

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

Nested If Exercise: game_level.c

- A player enters **y** or **n** to indicate whether they completed a level.
- Players who complete the level then enter how many coins they collected.
- Collecting all 10 coins earns 3 stars, while completing the level with fewer coins earns 1 star.
- A player who has not completed the level should be told to keep playing.

$ ./game_level
Did you complete the level? (y/n): **y**
How many coins did you collect? **10**
Bonus: 3 stars!

$ **./game_level**
Did you complete the level? (y/n): **n**
Keep playing!

Logical Operators

Often we want to combine more than one true/false condition

- For example
  - Is x greater than y and is x less than z?

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

Quick Questions: Logical Operators

true (1) or false (0)

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

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

Coding Exercise with Logical Operators

`character_cases.c`

- Write a program that:
  - Allows a user to enter a character
  - Prints whether it is an
    - uppercase letter,
    - lowercase letter
    - not a letter.

What **test cases** should we make sure we check?

Logical Operators Tips

For checking if a value is within a range

- Use a condition like this:

```
x > 0 && x < 10
```

NOT like this

```
0 < x < 10
```

- It is valid in maths
- It will compile and run in C
- But it is **not** checking whether x is greater than 0 and less than 10 in C

Logical Operators Tips

Beginners sometimes get confused with && and ||

```
((x > 0) || (x < 10))
```

- This is always true

```
((x < 0) && (x > 10))
```

- This is always false

Breaking Things

It is good to think about how someone can break your code

- What can go wrong with different inputs?
- It is important to have good error messages:
  - Tells the user exactly what has gone wrong
  - What they should do next

How can we check for input errors from scanf?

scanf return value and error checking

- We tell scanf what we want it to read
  - 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 successfully read
  - We can use this for error checking.

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

Coding Example

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

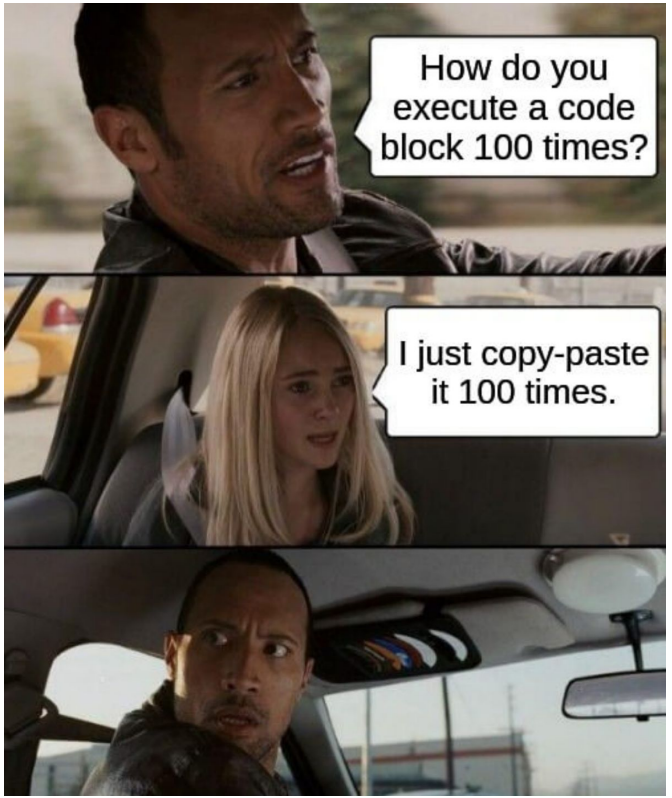
while bowl of chips is not empty
    eat a chip

While Loops

while in lecture
    stay awake

Repetition, Repetition, Repetition in C

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



While Loops

- A **`while`** loop repeats code **while a condition is true**
- The condition is checked **before each repetition**

```
while (condition) {
    // repeat this code
}
// continue running here when the condition is false
// and the loop has ended
```

Infinite while Loops

- It is very easy to make an infinite while loop
  - Handy Tip: **Press 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 < 10) {
    printf("You have done %d push-ups!\n", push_ups);
}
```

Controlling Loops

- We need to make sure our loops will finish!
  - We need to make sure our condition becomes false
  - We often do this by updating a counter

```c
int push_ups = 0;
while (push_ups < 10) {
    printf("You have done %d push-ups!\n", push_ups);
    push_ups = push_ups + 1;
}
```

3 Common 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
- **Sentinel loops**
  - A sentinel loop continues to execute until a special value (the sentinel value) is encountered in the user input.
- **Conditional loops**
  - They repeat while a condition is true
  - Useful when we do not know in advance how many times the loop will run

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.

```c
// 1. Initialise loop counter before the loop
int counter = 0;
while (counter < 5) { // 2. check loop counter condition
   printf("Here we go loop de loop!\n");
   counter = counter + 1; // 3. update loop counter
}
```

Sentinel Loops

- Process data until reaching a special value (sentinel value) in the user input

```c
char grade;
printf("Enter a grade (q to quit): ");
scanf(" %c", &grade);
// Stop when we reach the sentinel value 'q'
while (grade != 'q') {
    printf("You entered %c\n", grade);
    printf("Enter a grade (q to quit): ");
    scanf(" %c", &grade);
}
```

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("Kombucha target reached %d!!\n", total_kombucha_ml);
```

Conditional Loops

```c
int pin;
int confirmation;
int pins_match = 0;

while (pins_match == 0) {
    printf("Enter a PIN: ");
    scanf("%d", &pin);
    printf("Enter the PIN again: ");
    scanf("%d", &confirmation);
    if (pin == confirmation) {
        pins_match = 1;
    }
}
printf("PIN saved!\n");
```

- This conditional loop uses a flag variable.
  - A flag variable keeps track of whether a condition has been met.
  - In this example the flag variable is pins_match. It is using 0 for false and 1 for true.

Code Demo

`while_infinite.c`

`while_count.c`

`while_sentinel.c`

`while_condition.c`

`while_condition_flag.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/t2VfYYbXmR

What did we learn today?

- Recap and Constants
- 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_sentinel.c,
  - while_conditional.c, while_conditional_flag.c, while_scanf_sum.c

Reach Out

Content Related Questions:

Course Forum

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

Student Support - I Need Help With...

My Feelings and Mental Health Feeling depressed, overwhelmed or not your usual self?	**Mental Health Connect** student.unsw.edu.au/**mhc** **TalkCampus** student.unsw.edu.au/**Talkcampus**	**UNSW (24/7) Mental Health Support:** +61(2) 9385 5418 **Text support After Hours**: 0485 826 595 **Offshore** **Call 24-hour Medibank Hotline:** +61 2 8905 0307
Uni and Life in Australia Stress, financial, visas, Accommodation & More	**Student Support** **Indigenous Student Support**	student.unsw.edu.au/**advisors** nura-gili-centre-indigenous-programs
Reporting Sexual Assault/Harassment	**Equity Diversity & Inclusion (EDI)**	edi.unsw.edu.au/**sexual-misconduct**
Educational Adjustments To manage my studies and disability / health condition	**Equitable Learning Services (ELS)**	student.unsw.edu.au/**els**
Academic and Study Skills	**Academic Skills**	student.unsw.edu.au/**skills**
Special Consideration Because life impacts our studies and exams	**Special Consideration**	student.unsw.edu.au/**special-consideration**
Physical Health **Doctors visits, Vaccinations**	**Health Service**	student.unsw.edu.au/**hsu**