COMP1511/1911 Programming Fundamentals

Week 2 Lecture 2

Loops
Custom Data Types

Yesterday's Lecture

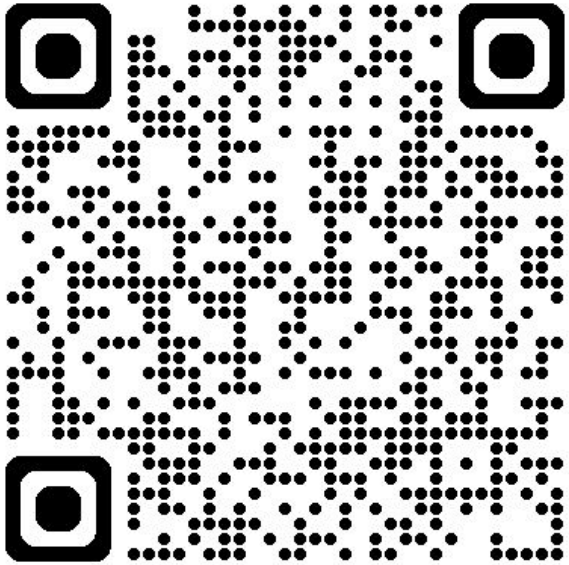
- **Conditions and if statements**
  - Relational Operators: `<`, `<=`, `>`, `>=`, `!=`
  - Logical Operators: `&&`, `||`, `!`
  - If-else, else-if chains and nested if statements
  - Error checking with scanf
- **While loops**
  - Infinite loops

Today's Lecture

- **While loops**
  - Infinite Loops
  - Counting loops
  - Sentinel loops
  - Conditional Loops
- **Nested While Loops**
  - Loop over 2 dimensions!
- **Custom data types**
  - Create our own types to give data more structure and meaning
    - struct
    - enum

Link to Week 2 Live Lecture Code

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



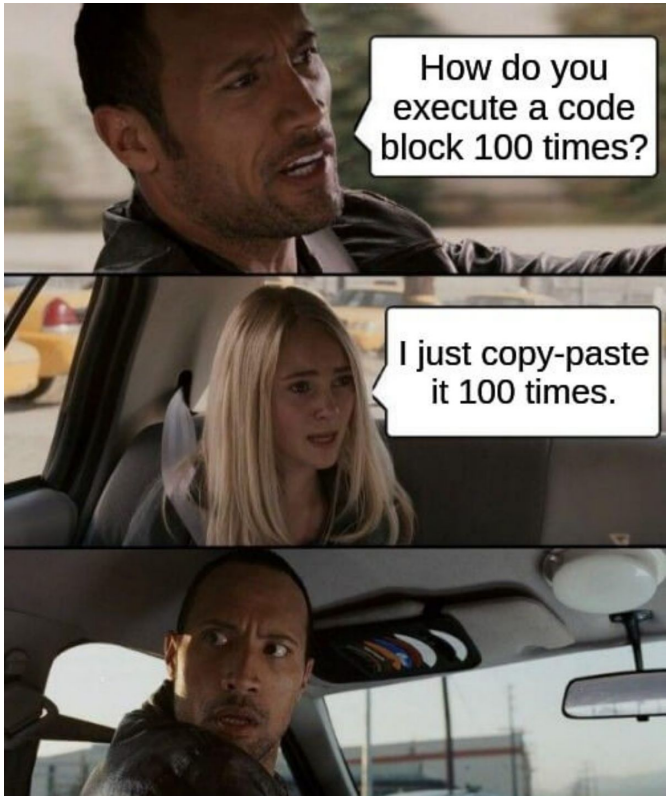
```
while bowl of chips is not empty
    eat a chip
```

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);
}
```

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

Code Demo

`while_infinite.c`

`while_count.c`

`while_sentinel.c`

`while_conditional.c`

`while_conditional_flag.c`

Exercise

- Write code to read in an int value n and print n * on one line

  `while_stars.c`
- Write a program that reads integers from the user until a non-integer is entered

  `while_scanf.c`

  .

Recap: scanf Return Value

```c
int value;
int n_read;
printf("Enter an integer: ");
n_read = scanf("%d", &value);
```

What would the value of n_read be if the user typed in:

5

potato 3

Quick Break. Back Soon…

Nested Loops

How could we print out something like this?

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Or this?

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Or even this?

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

We can have a loop inside a loop!

Code Demo: Nested While Loops

`grid_stars.c`

`numbers_grid.c`

`triangle.c`

Nested Loops

- A loop in a loop
- Each time the outer loop repeats:
  - the inner loop runs through all of its repetitions
- The inside loop ends up running a LOT of times
  - How many times does the second hand go around the clock or every minute? For every hour?

Custom Data Types

Custom Data Types

So far, we have used built-in C data types (int, char, double)

We will look at 2 different ways we can create Custom Data types.

struct: create a type to store a collection of related values
enum: create a type with a set of named values

With custom data types, we design what we can store in variables of these types.

Organising related data

Once we have lots of variables, we need ways to organise related data

Organising related data

Are these variables related to each other in any meaningful way?

```
char my_first_initial = 'A';
char my_last_initial = 'F';
int my_age = 23;
double my_lab_mark = 2.4;
char brianna_first_initial = 'B';
char brianna_last_initial = 'K';
int brianna_age = 21;
double brianna_lab_mark = 9.9;
```

Organising related data

Are these variables related to each other in any way?

```
char my_first_initial = 'A';
char my_last_initial = 'F';
int my_age = 23;
double my_lab_mark = 2.4;
```

```
char brianna_first_initial = 'B';
char brianna_last_initial = 'K';
int brianna_age = 21;
double brianna_lab_mark = 9.9;
```

We could group the data related to each person.

But how can we do this in C?

Organising related data

What about this data?

```
int x1 = 0;
int y1 = 0;
int z1 = 0;
int x2 = 10;
int y2 = -5;
int z2 = 5;
```

Organising related data

```
int x1 = 0;
int y1 = 0;
int z1 = 0;
int x2 = 10;
int y2 = -5;
int z2 = 5;
```

We could group the data related to the coordinates of each point

But how can we do this in C?

User Defined Type: struct

- `struct`s allow us to define our own data types (structures) group **related values of different types** together
- Before we can create struct variables, we need to define the struct
  - This does not create a variable or set aside memory
  - It defines the **blueprint** for the type.
- Then we can declare and use variables of that struct type

1. Defining a struct

- We define our structs before our main function.
- **`struct`**s are types that we design, made up of data that belongs together
  - We call the elements **members** or **fields**
  - Each member has a **type** and a **name**

```
struct student {
    char first_initial;
    char last_initial;
    int age;
    double lab_mark;
};
```

2. Declaring a struct variable

- Creating variables using your custom **`struct`** type

```c
struct student {
    char first_initial;
    char last_initial;
    int age;
    double lab_mark;
};
```

```c
int main(void) {
    // Declare a variable
    // of type struct student
    struct student brianna;
```

3. Accessing and assigning struct members

- We access a member of a struct by using the dot operator `.`

```c
struct student {
    char first_initial;
    char last_initial;
    int age;
    double lab_mark;
};
```

```c
int main(void) {
    // Declare a variable
    // of type struct student
    struct student brianna;
    // Assign values to
    // struct members
    brianna.first_initial = 'B';
    brianna.last_initial = 'K';
    brianna.age = 21;
    brianna.lab_mark = 9.9;
```

Exercise: Using structs

- Increment the age field
- Read in updated lab mark from the user.
- Print out updated student data

```c
struct student {
    char first_initial;
    char last_initial;
    int age;
    double lab_mark;
};
```

```c
int main(void) {
    // Declare a variable
    // of type struct student
    struct student brianna;
    // Assign values to
    // struct members
    brianna.first_initial = 'B';
    brianna.last_initial = 'K';
    brianna.age = 21;
    brianna.lab_mark = 9.9;
```

Exercise: Using structs

- Enter data for two points
- Print out both points

```c
struct point {
    int x;
    int y;
    int z;
};
```

```c
int main(void) {
    // Declare 2 variables of
    // type struct point
    struct point point_1;
    struct point point_2;
```

Enumerations: enum

An **enum** defines a group of named integer constants that we use as the meaningful values for a type.

// Defines the set of possible values for the type enum direction

```
enum direction {NORTH, SOUTH, EAST, WEST};
```

We use a variable of that enum type to store one of these named values.

// Create a variable of type enum direction and intitialise it

```
enum direction my_dir = SOUTH;
```

Enumerations

- Enums give meaningful names to integer values, making code easier to read and maintain
  - By default the enumerated constants will have int values 0, 1, 2, …
  - You can also define the values yourself
  - Note you can’t have two enums with the same constant names

```
// E.g. define an enum for day of the week
enum weekdays {MON, TUE, WED, THU, FRI, SAT, SUN};

// E.g. define an enum with specified int values
enum status_code {OK = 200, NOT_FOUND = 404};
```

enum code example

```c
// Note: We will define enum types before the main
// Define an enum with days of the week
// MON will have value 0, TUE 1, WED 2, etc
enum weekdays {MON, TUE, WED, THU, FRI, SAT, SUN};

int main (void) {
    enum weekdays day = SAT;
    // This will print out 5
    printf("The day number is %d\n", day);
    return 0;
}
```

enum vs #define

- **Advantages of enums:**
  - Define a specific fixed set of related constants
  - Helps catch mixing values from different enums
    - e.g. using **`FEB`** from **`enum month`** as a **`enum weekday`**
  - Less repetitive and more compact
  - Values can be numbered automatically
- **#define is still useful for:**
  - Constants that are not integers
  - Stand alone constant values

Putting it Together: structs with enums

We can have enum members in our structs!

```
enum direction {
    NORTH, SOUTH, EAST, WEST
};

struct player {
    enum direction current_dir;
    int score;
};
```

```
struct player hero;
hero.current_dir = EAST;
hero.score = 0;
```

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/F56gV5WHM7

What did we learn today?

- While loops
  - `While_count.c, while_sentinel.c, while_conditional.c, while_conditional_flag.c while_stars.c while_scanf.c`
- Nested while loops
  - `grid_stars.c, numbers_grid.c, pyramid.c`
- structs
  - `struct_student.c, struct_points.c`
- enums
  - `enum_weekdays.c`

Next week

- Putting it together:
  - Pokemon struct that contains enums!!
- Functions
- Style

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**