COMP1511/1911 Programming Fundamentals

Week 2 Lecture 2

Loops
Custom Data Types

Monday's Lecture

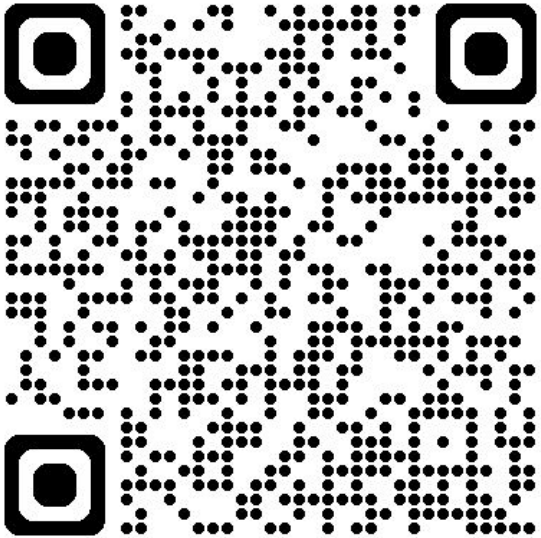
- Conditions
  - Relational Operators
  - Logical Operators
- If statements
  - if-else
  - chaining if-else
- We did not complete the following which we will do today!
  - Nested If Statements
  - While loops

Today's Lecture

- From Monday
  - nested if statements
  - while loops
- Nested While Loops
- Custom data types
  - structs
  - enums (maybe if we get time)

Link to Week 2 Live Lecture Code

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



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");
}
```

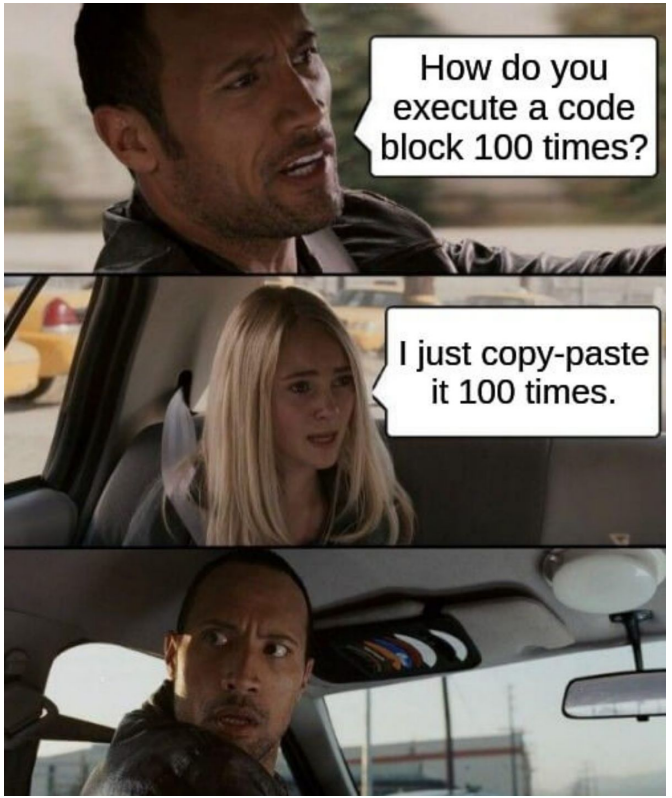
while bowl of chips is not empty
    each chip

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`

Quick Break

Nested While Loops

- A loop in a loop
- If we put a loop inside a loop . . .
- Each time a loop runs
  - It runs the other loop
- The inside loop ends up running a LOT of times
  - How many times does the second hand go around the clock for every minute? For every hour?

Why are nested while loops useful?

How could we print out something like 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 this?

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

Or this?

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

Code Demo Nested While Loop

`grid.c`

`pyramid.c`

`pattern.c` (exercise you can try and we will do on Monday)

`clock.c`

Custom Data Types

Organising related data

Is there a better way of storing related data?

```
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;
```

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

Organising related data

Is there a better way of storing related data?

```
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;
```

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

User defined Data Type: struct

- So far, we have used built-in C data types (int, char, double)
- These store a single item of that type
- `struct`s allow us to define our own data types (structures) to store a collection of types

User defined Data Type: struct

To create a struct, there are three steps:

1. Define the struct (outside the main)
    - Note this does not create a variable or set aside any memory.
    - It just defines the type.
2. Declare the struct variable/s
3. Initialise the struct variables/s

1. Defining a struct

- We define our structs before our main function.
- `structs` are types that we design, made up of data elements that we decide belong together
  - we call these elements **members** or **fields**
  - we need to define a type and name for each member

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

```
struct coordinate {
    int x;
    int y;
};
```

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

```c
struct coordinate {
    int x;
    int y;
};

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

3. Initialising struct data

- 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;
    // Initialise the members of
    // your struct variable
    brianna.first_initial = 'B';
    brianna.last_initial = 'K';
    brianna.age = 21;
    brianna.lab_mark = 9.9;
```

Quick Question: struct

- How would we set point_1 to (0, 0) and point_2 to (10, -5)

```c
struct coordinate {
    int x;
    int y;
};
```

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

Quick Question Solution: struct

- How would we set point_1 to (0, 0) and point_2 to (10, -5)

```
struct coordinate {
    int x;
    int y;
};
```

```
int main(void) {
    // Declare 2 variables of
    // type struct coordinate
    struct coordinate point_1;
    struct coordinate point_2;
    point_1.x = 0;
    point_1.y = 0;
    point_2.x = 10;
    point_2.y = -5;
```

Enumerations

- Data types that allow you to assign names to integer constants to make it easier to read and maintain your code
  - By default the enumerated constants will have int values 0, 1, 2, …
  - Note you can’t have two enums with the same constant names

```
// Example of the syntax used to define an enum
enum enum_name {STATE0, STATE1, STATE2, ...};

// 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
// Define an enum with days of the week
// make sure it is outside and before the main function
// 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;
    day = SAT;
    // This will print out 5
    printf("The day number is %d\n", day);
    return 0;
}
```

enum vs #define

- enums are useful when we want to define a specific fixed set of constants
- The advantages of using enums over #defines
  - Enumerations are automatically assigned values, which makes the code easier to read
    - Think of the case where you have a large number of related constants
- #define are useful for other contexts such as constants that are not integers or stand alone constant values

struct with enum members!

- We can have enum members in our structs!

```
enum student_status {
    ENROLLED, WITHDRAWN, LEAVE
};

struct student {
    enum student_status status;
    double wam;
};
```

```
struct student z123456;
z123456.status = ENROLLED;
z123456.wam = 95.9;
```

Coding Example:

`pokemon.c`

- We can have enum members in our structs!
- Create a enum for pokemon types FAIRY, WATER, FIRE etc
- Create a struct called pokemon with a field for the type and some other relevant fields
- Make a pokemon variable and set it with data

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/Pc7gMkscYy

What did we learn today?

- Nested if statements
  - `nested.c`
- While loops
  - `while_infinite.c`, `while_count.c`, `while_conditional.c`, `while_sentinel.c`, `while_scanf_sum.c`
- Nested while loops
  - `grid.c`, `pyramid.c`, `pattern.c`, `clock.c`

What did we learn today?

Custom Data types

- structs
  - `struct_student.c, struct_points.c`
- enums
  - `enum_weekdays.c`
- structs containing enums
  - `pokemon.c`

Reach Out

Content Related Questions:
Forum

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