COMP1511/1911 Programming Fundamentals

Week 2
Lecture 2

Previously On...

- Basic if statements
  - Running our code based on certain conditions
  - Decision problems
  - Relational operators
  - Logical operators
- While loops
  - Repeating code without repeating code
  - Three main kinds
    - Counting loop
    - Sentinel loop
    - Conditional loop

Today

- Refresher on ifs and loops
  - Can be challenging, don’t get discouraged!
- Loops within loops
  - Loops within loops within loops
- Custom data types
  - Structs
  - Enums

Where is the Code?

- Lecture code is updated every hour on the hour
- Live lecture code can be found here
  - https://cgi.cse.unsw.edu.au/~cs1511/26T2/

Why Bother?

- While loops and if statements can be tricky
  - So let’s refresh them!
  - Repetition is vital to ensuring fluency
- Sometimes we have problems within problems
  - Solved with loops within loops
- Structs and enums allow us to more neatly organise our code

Refresher if statements

- if statements answer binary yes or no questions
  - Not “tea or coffee?”
- However, we can chain them together to ask more complex questions
  - “Would you like tea?” || “Would you like coffee?”
    - Wouldn’t tell us which one, just if they want one, or both!

Refresher while Loops

- Repeating tasks doesn’t require copying code
- while loops allow us to repeat code based on a condition
  - Similar to the questions we ask in an if statement
    - if this condition is true, run the code, and check the condition again

Infinite while Loops

- It’s very easy to make code that goes on forever
  - If our loop condition is never checked or updated, our while loop will never end

```c
#include <stdio.h>

int main(void) {
    int answer = 12;
    // When will this loop end? HINT - Never
    while (answer == 12) {
        printf("This will take a while\n");
    }
    return 0;
}
```

Three Main while Loops

- Count loop
  - Repeats a set number of times
- Sentinel loops
  - Repeats until a flag is flipped
- Conditional loops
  - Repeats until a condition is met

Count Loops

- Use a variable to count how many times a loop runs
  - int declared before the loop
- The “termination condition” can be checked in the while loop
- It will be updated inside the loop

```c
#include <stdio.h>

int main(void) {
    int counter = 1;
    while (counter < 5) {
        printf("Printed %d times\n", counter);
        counter = counter + 1
    }
    return 0;
}
```

Sentinel Loops

- Count loops depend on us knowing how many times we want something to happen
  - What if we want something to happen until a certain event?
    - E.g keep prompting for input until an odd number is entered
      - We don’t know how long this will take, but we will know when it’s happened
- A sentinel is a flag value
  - It tells the loop when to stop

Sentinel Loops

- Sentinel loops can also use a variable to decide to exit the loop at any time
- We call this variable a “sentinel”
  - Like an on/off switch for the loop
  - Declared and set outside the loop
  - Its termination condition can be checked in the while condition
  - It will be updated inside the loop
    - Often attached to a decision statement

Sentinel Loops

```c
#include <stdio.h>

int main(void) {
    int exit_condition = 0;
    int coconuts_harvested = 0;
    int total_coconuts = 0;
    while (exit_condition == 0) {
        printf("How many coconuts will you collect? ");
        scanf("%d", &coconuts_harvested);
        total_coconuts = total_coconuts + coconuts_harvested;
        if (total_coconuts > 40) {
            printf("We have enough food to survive! Yay!!\n");
            exit_condition = 1;
        }
        else {
            printf("Let's keep harvesting!\n");
        }
    }
    return 0;
}
```

Conditional Loops

- Use a condition to decide when the exit the loop
  - We also don’t know how many times something will repeat, but we are slowly making progress towards the condition
- We will terminate as a result of some kind of calculation

Conditional Loops

```c
#include <stdio.h>

int main(void) {
    int sum = 0;
    int adder = 0;
    while (sum < 100) {
        printf("Please enter a number: ");
        scanf("%d", &adder);
        sum = sum + adder;
    }
    return 0;
}
```

while Inside a while

- If we put a loop inside a loop
  - Each time the outside loop runs
    - The inside loop is run too
- The inside loops runs a LOT of times

Print Out a Grid of Numbers

- We want to print out the following grid
  - 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
- Extension
  - Instead of 1-5 5 times, we do 1-n n times, where n is a number inputted by the user

Print Out a Grid of Numbers

- Think of the grid not ass numbers, but as 5 rows of bricks
  - To build it, you must lay each brick one after the other side-by-side to finish the row
  - You must finish the row before you can start the next one
- So we start at Row 1
  - Lay brick 1, brick 2, brick 3, brick 4, brick 5
    - Now that we hit 5, we need to physically move to the next row
- Repeat for Row 2, and keep repeating until Row 5 is done

Print Out a Pyramid of Numbers

- What if we want to print out a half pyramid of numbers?
  - ```
    1
    1 2
    1 2 3
    1 2 3 4
    1 2 3 4 5
    ```
- Let’s break down the problem
    ```

More while Loop Fun

- Re-prompting the user if they enter the wrong number
- Extending pyramid and grid printing to allow user input

Please Practice!

- The concepts introduced this week are imperative to understanding programming
  - Programming is a topic that builds on itself
    - You cannot build a house if the foundation is wobbly!
-
```
if (understanding == 0) {
    printf(“Please practice!\n”);
}
else {
    printf(“You should probably still practice!\n”);
}
```

Structs

- Organising different types into one related whole
- Structs (short for structures) are a way to create custom variables
- Structs are variables made up of other variables
  - If you’re familiar with objects in other languages, it’s not really like that, it’s an entirely new thing!

Why?

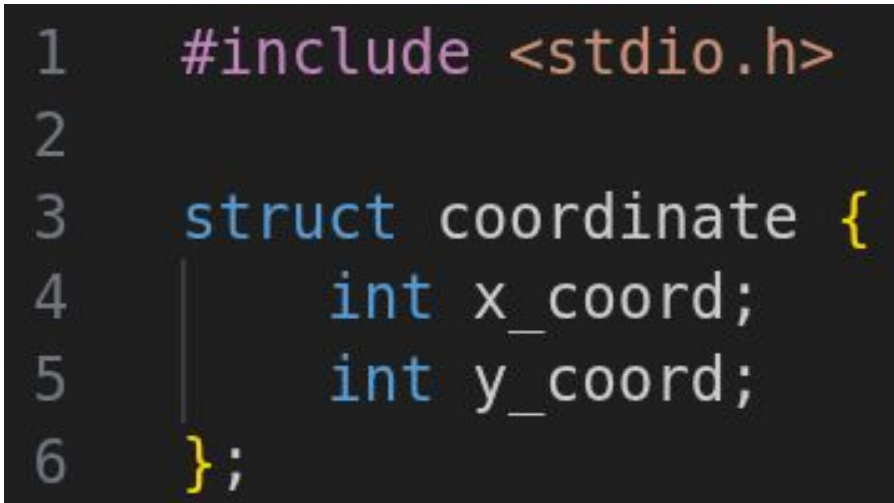
- Helps us organise related but different components
- Useful in defining real life problems
  - What are some real life things that are made up of a bunch of smaller components?
    - Cars
      - Make, Model, Year
    - People
      - Name, Age, Height
    - Survivor seasons
      - Number, Title, Length, Rating

Creating Structs

- There are three steps
  1. Define the struct (outside the main)
  2. Declare the struct (inside the main)
  3. Initialise the struct (inside the main)

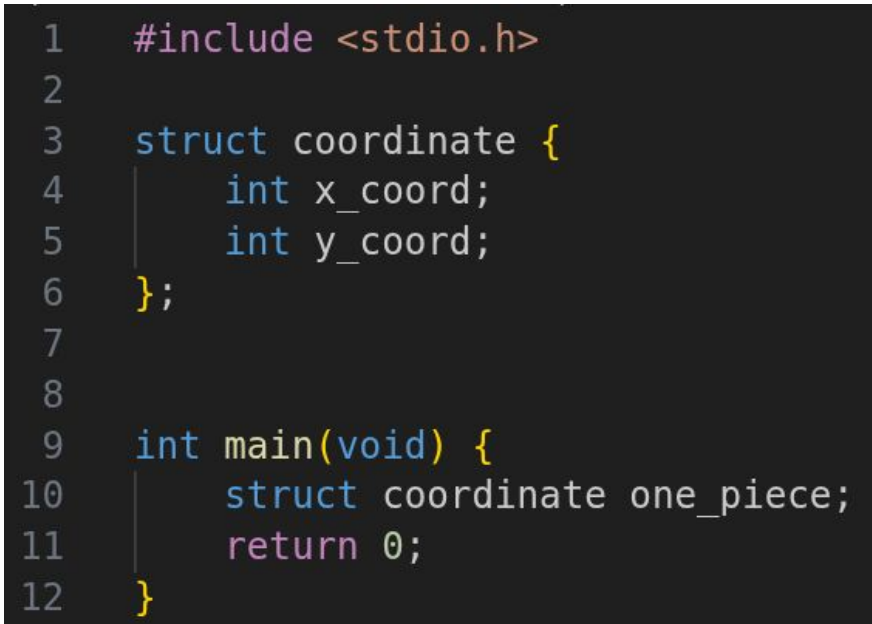
Defining the Struct

- Structs are made up of things we decide
  - Therefore, we need to declare them, so C knows what we're working with
- Define before our main function using special syntax



Declaring the Struct

- To declare the struct inside your main function (or wherever you're using it) you must use the following syntax
  - Note that you need to let C know that you're using a struct



```c
#include <stdio.h>

struct coordinate {
    int x_coord;
    int y_coord;
};


int main(void) {
    struct coordinate one_piece;
    return 0;
}
```

Initialise the Struct

- We access a struct member using the dot operator
  - This is how we both declare the values, and use them later on

```c
#include <stdio.h>

struct coordinate {
    int x_coord;
    int y_coord;
};


int main(void) {
    struct coordinate one_piece;
    one_piece.x_coord = 15;
    one_piece.y_coord = 100;
    printf("The One Piece is located at X = %d, and Y = %d!\n", one_piece.x_coord, one_piece.y_coord);
    return 0;
}
```

Using Structs

- We can also update struct values after initialisation

```c
#include <stdio.h>

struct coordinate {
    int x_coord;
    int y_coord;
};


int main(void) {
    struct coordinate one_piece;
    one_piece.x_coord = 15;
    one_piece.y_coord = 100;
    one_piece.x_coord = one_piece.x_coord * 10;
    one_piece.y_coord = one_piece.x_coord + 100;
    printf("The One Piece is located at X = %d, and Y = %d!\n", one_piece.x_coord, one_piece.y_coord);
    return 0;
}
```

Enums

- enums (short for enumerations) are integer data types you created with a limited range of constants
- Used to assign names to integer constants
  - The name makes the program easier to maintain

```c
#include <stdio.h>

enum enum_name {STATE_1, STATE_2, STATE_3};

enum weekdays {MON, TUE, WED, THURS, FRI, SAT, SUN};

enum state_flag {SUCCESS = 1, FAILURE = 2};
```

Enums Example

- Note that enums start counting at 0
  - This will print 5

```c
#include <stdio.h>

enum weekdays {MON, TUE, WED, THURS, FRI, SAT, SUN};

int main(void) {
    enum weekdays day;
    day = SAT;
    printf("The day of the week is %d\n", SAT);
    return 0;
}
```

Enums Example

- What if we want to start at 1, or 2, or 3, or n?
  - If you set an initial value, it will enumerate from there

```c
#include <stdio.h>

enum survivor {BORNEO = 1, THE_AUSTRALIAN_OUTBACK, AFRICA, MARQUESAS, THAILND, AMAZON, PEARL_ISLANDS, ALL_STARS,
               VANUATU, PALAU, GUATEMALA, PANAMA, COOK_ISLANDS, FIJI, CHINA, MICRONESIA, GABON, TOCANTINS, SAMOA,
               HEROES_VS_VILLAINS, NICARAGUA, REDEMPTIONS_ISLAND, SOUTH_PACIFIC, ONE_WORLD, PHILIPPINES, CARAMOAN,
               BLOOD_VS_WATER, CAGAYAN, SAN_JUAN_DEL_SUR, WORLDS_APART, CAMBODIA, KAOH_RONG, MILLENIALS_VS_GEN_X,
               GAME_CHANGERS, HEROES_VS_HEALERS_VS_HUSTLERS, GHOST_ISLAND, DAVID_VS_GOLIATH, EDGE_OF_EXTINCTION,
               ISLAND_OF_THE_IDOLS, WINNERS_AT_WAR, _41, _42, _43, _44, _45, _46, _47, _48, _49, _50_IN_THE_HANDS_OF_THE_FANS};

int main(void) {
    enum survivor season;
    season = GABON;
    printf("Survivor Season %d is good!\n", season);
    return 0;
}
```

enum vs #define

- Enumerations follow scope rules
  - You can’t use the same enum name in two different types of enums
- Enumerators are automatically assigned
  - Makes code easier to read
    - What if you have a large number of constants?
- We use enums when we want variables to have a specific set of values

More struct and enum Fun

- Let’s play around with structs and enums!

What We Learnt

- Refreshing if and while
- structs and enums
  - User defined variables and constants

Reach Out

- Check the course forum for questions!
  - Post there too!
- Admin questions
  - cs1511@unsw.edu.au

Thank You

Questions?