COMP1511 - Programming Fundamentals

Week 4 - Lecture 7

Functions in Code

```
// a function declaration
int add (int a, int b);   a+b.

int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    // use the function here
    int total = add(firstNumber, secondNumber);
    return 0;
}

// the function is defined here
int add (int a, int b) {        ?
    return a + b;
}
```

Why use functions?

Why do we separate code into functions?

Saves us from repeating code

- Instead of replicating code, we can write it once
- This also makes the code much easier to modify 改

Easier to organise code

- Complex functionality can be hidden inside a function
- The flow of the program can be read easily with clear function names

C Libraries

We've already used `stdio.h` several times

- C has other standard libraries that we can make use of
- The simple C reference in the Weekly Tests has some information
- `math.h` is a useful library of common maths functions
- `stdlib.h` has some useful functions
- Look through the references (including `man` manuals in linux)
- Don't worry if you don't understand the functions yet, some of them have no context in the programming we've done so far

Using Libraries

```
// include some libraries 库
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

int main (void) {
    int firstNumber = -4;
    int secondNumber = 6;

    // change a number to its absolute value
    firstNumber = abs(firstNumber);

    // calculate a square root
    int squareRoot = sqrt(firstnumber);
    printf("The final number is: %d", squareRoot);
    return 0;
}
```

Recap of Arrays

A collection of variables

- Contains multiple variables all of the same type
- Declared using a variable type and a size
- Individual variables are accessed using an index

Indexes	0	1	2	3	4
An Array	63	88	43	55	67

Using Arrays in C

Some example code of an array

```
int main (void) {
    // declare an array of doubles, size 4, initially all 0
    double myArray[4] = {0};

    // assign a value
    myArray[1] = 0.95;
    // test a value
    if (myArray[2] < 1) {
        // print out a value
        printf("Third element is: %lf", myArray[2]);
    }
```

Accessing multiple values at once

Loops and Arrays go together perfectly

- Accessing all members is a reasonably simple while loop

```
int main (void) {
    // declare an array of doubles, size 4, initially all 0
    double myArray[4] = {0};

    // loop through the array and output the elements
    int i = 0;
    while (i < 4) {
        printf("%lf\n", myArray[i]);
        i++;
    }
```

Creating Arrays with certain sizes

Arrays start at an exact size and don't change

- When we create an array, we give it a size and a type
- Both of those are fixed and won't change

```c
int main (void) {
    // declare an array of doubles,
    // size 4
    double myArray[4] = {0};

}
```

```c
int main (void) {
    // This declaration is not
    // possible!
    int arraySize = 4;
    double myArray[arraySize] = {0};

}
```

We can't declare an array with a variable size like this!

Using Constants for Array Sizes

If we do want to be able to change the size in code . . .

- We can use a constant to set the size
- Unlike a variable, this cannot change after it is compiled
- It does make our lives much easier if we need a change mid-project

```
#define ARRAY_SIZE 4

int main (void) {
    // This declaration allows us to change the
    // array size while coding
    double myArray[ARRAY_SIZE] = {0};
}
```

Two Dimensional Arrays

Arrays inside arrays

- Can be thought of like a grid 网格
- The outer array contains arrays
- Each array is a row of the grid
- Addressed using a pair of integers like coordinates
- All inner arrays are of the same type

Indexes	0	1	2	3	4
0	63	88	43	55	67
1	54	52	91	21	32
2	77	58	1	61	79

A 2D Array

[1][3]

Two Dimensional Arrays in Code

```
int main (void) {
    // declare a 2D Array
    int grid[4][4] = {0};

    // assign a value
    grid[1][3] = 3;
    // test a value
    if (grid[2][0] < 1) {
        // print out a value
        printf("The bottom left square is: %d", grid[3][0]);
    }
```

Let’s work with 2D Arrays

I would like to make a simple game called “The Tourist”

- The world is a square grid
- The tourist can move up, down, left or right
- Be able to print out the world, including the location of the tourist
- The tourist likes seeing new things . . .
- Track where they've been
- And lose the game if we revisit somewhere we've been

Print Map

start code

Here's a handy function that we'll be reusing

```
void printMap(int map[N_ROWS][N_COLS], int posR, int posC) {
    int row = 0;
    while (row < N_ROWS) {
        int col = 0;
        while (col < N_COLS) {
            if (posR == row && posC == col) {
                printf("T ");
            } else {
                printf("%d ", map[row][col]);
            }
            col++;
        }
        row++;
        printf("\n");
    }
}
```

The Square Grid World

Break down into parts

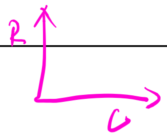
Variables for the grid and the tourist's position

```c
#include <stdio.h>

// The dimensions of the map
#define N_ROWS 10
#define N_COLS 10

int main (void) {
     int map[N_ROWS][N_COLS] = {0};
     int posR = 0, posC = 0;
```

empty



Controlling the Tourist

Next Steps

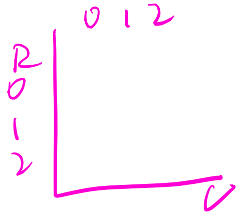
- Let’s add movement
- Then track where the Tourist has been, using the map
- After that, we'll check for places we've already been

Looping

- We can loop repeatedly for “turns” to allow the user to input directions

Movement - this code will loop

```
printf("Please enter a numpad direction or 0 to exit: ");
int input;
scanf("%d", &input);
if (input == 4) {
    posC--; ←
} else if (input == 8) {
    posR--; ↑
} else if (input == 6) {
    posC++; →
} else if (input == 2) {
    posR++; ↓
} else if (input == 0) {
    exit = 1;
} else {
    printf("Input is not a numpad direction, please use 2,4,6 or 8\n");
}
```



Tracking the Tourist using the Map

Set each location we visit to 1

```
// loop and let the user control the Tourist's movement
int exit = 0;→ exit loop
while (!exit) {
    // mark the location as having been visited by incrementing递增
    map[posR][posC] = 1;

    // show the current status
    printMap(grid, posR, posC);

    printf("Please enter a numpad direction or 0 to exit: ");

    // Movement code from previous slide goes here . . .
```

1 isn't as helpful as "EXPLORED"

Let's swap out the number for a more readable #define

```c
#include <stdio.h>

// The dimensions of the map
#define N_ROWS 10
#define N_COLS 10

// Has the square been explored before?
#define UNEXPLORED 0
#define EXPLORED 1
```

未探索

已探索

Have we been here before?

We want the game to end if the tourist revisits a location

- If the location we visit is already 1
- Then we're going to exit the game
- We can add this check after our movement

```
// Check if we've been here before
if (map[posR][posC] == 1) {
    printf("We've already been here! How boring!\n");
    exit = 1;
}
```

The Tourist Game

This is now roughly complete

- We can move the tourist
- We can track where we’ve been
- We can display where we've been as well as current location
- We can exit if we revisit a location

But how safe is it?

- Try different inputs
- Try moving around a bit

Walking off the edge of the map

Our Tourist can walk outside of the bounds of our arrays!

Let's add some code to check if we're outside the map and stop that movement

```
// Check if we've walked off the map
if (posR < 0) {
    posR = 0;                 10.
} else if (posR >= N_ROWS) {
    posR = N_ROWS - 1;
}
if (posC < 0) {
    posC = 0;                 10.
} else if (posC >= N_COLS) {
    posC = N_COLS - 1;
}
```