COMP1511/1911 Programming Fundamentals

Week 4
Lecture 2

Assignment 1

- Released yesterday!
- Aims of the assignment
  - Apply arrays and two-dimensional arrays in problem solving
  - Apply good style to your code
  - Apply the use of functions in code
  - Practice skills in debugging code
    - and skills in hunting for missing semicolons
      - Hopefully my lectures have taught you well

Previously On...

- Recap strings
- Recap arrays
- Array of collections
  - Array of structs
  - Array of arrays

Today

- Recap 2D arrays
- Recap strings.h
- Command line arguments

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?

- 2D arrays are very important for the assignment
  - We want to do well on the assignment
- 2D arrays are also very powerful representations of the real world
  - Maps
  - Board games
- strings.h is a new library and practice makes perfect
- Command line arguments let us get input before our program even begins

Array of Arrays Recap

- Let’s say we declare an arrays
  - `int array[3][4]`
- It looks like this

Array of Arrays Recap

- Let’s say we declare an arrays
  - `int array[3][4]`
- It looks like this

array[0][0]	array[0][1]	array[0][2]	array[0][3]
array[1][0]	array[1][1]	array[1][2]	array[1][3]
array[2][0]	array[2][1]	array[2][2]	array[2][3]

Problem Time!

- Sum up each row in an array and output the max sum
- Decide whether the matrix is a lower triangular matrix
- Starting a simple tic-tac-toe

Break Time!

...probably

fgets

- fgets takes three arguments
  - The array the string will be stored in
  - The number of characters that will be read in
  - Where the string is coming from
    - In our case, always stdin

```c
#include <stdio.h>

#define MAX_SIZE 64

int main(void) {
    char array[MAX_SIZE];
    fgets(array, MAX_SIZE, stdin);
    printf("%s", array);
    return 0;
}

```

Reading in Repeatedly

- Reads in input until ctrl+d is entered

```c
#include <stdio.h>

#define MAX_SIZE 64

int main(void) {
    char array[MAX_SIZE];
    while (fgets(array, MAX_SIZE, stdin) != NULL) {
        printf("%s", array);
    }
    return 0;
}
```

fputs

- Function for displaying a string in the terminal
- Takes two arguments
  - The array to be outputted
  - Where the string will output to
    - In our case, always stdout

```c
#include <stdio.h>

#define MAX_SIZE 64

int main(void) {
    char array[MAX_SIZE];
    fgets(array, MAX_SIZE, stdin);
    fputs(array, stdout);
    return 0;
}
```

string.h

- We’re using a new library!
  - #include <string.h>
- Gives us access to more functions for strings
  - Similar to how stdio.h gives us functions for input and output
- Read more here:
  - https://www.w3schools.com/c/c_ref_string.php

Other String Functions

- strlen()
  - Gives us the length of a string
- strcpy()
  - Copies the contents of one string to another
- strcat()
  - Attaches a string to the end of another (concatenate)
- strcmp()
  - Compares two strings
- strchr()
  - Finds the first or last occurrence of a character

Example

- Using some string.h functions!

```c
#include <stdio.h>
#include <string.h>

int main(void) {
    char array[10];
    //Copy Survivor into our array
    strcpy(array, "Survivor");
    printf("%s\n", array);

    //Checks length of our string
    int length = strlen(array);
    printf("%d\n", length);

    //Compares two strings
    int compare_string = strcmp("Henry", "Epic");
    printf("%d\n", compare_string);

    int new_compare_string = strcmp("Survivor", array);
    printf("%d\n", new_compare_string);

    return 0;
}
```

Problem Time

- Implement our own strlen function that prints the number of characters in a string
- Take string input and print it in reverse
- Count the total number of words in a string that you read in from the user
- Count the total number of alphabetic characters, digit characters and special characters in a string that is read in from the user

Command Line Arguments

- So far we have only given input to our program after we run it
  - Therefore our main has taken no inputs
    - int main(void)
- Command line arguments allow us to give inputs to our program at the time that we start running it!

Command Line Arguments

- For example:

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("The command line arguemnt at index 0 (argv[0] is %s\n", argv[0]);
    printf("The command line arguemnt at index 1 (argv[1] is %s\n", argv[1]);
    printf("The command line arguemnt at index 2 (argv[2] is %s\n", argv[2]);
    printf("The command line arguemnt at index 3 (argv[3] is %s\n", argv[3]);
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

z3548950@vx16:~/public_html/COMP151126T2/code/week4$ ./prog survivor is cool
The command line arguemnt at index 0 (argv[0] is ./prog
The command line arguemnt at index 1 (argv[1] is survivor
The command line arguemnt at index 2 (argv[2] is is
The command line arguemnt at index 3 (argv[3] is cool
```

Breaking It Down

- `int argc`
  - Counter for how many command line arguments you have
    - Including your program name
- `char *argv[]`
  - Array of the different command line arguments (separated by a spaces)
    - Each command line argument is a string (an array of char)

Numbers Instead of Strings

- If you want numbers, you'll need to convert strings to integers
  - atoi() is a useful stdlib function to do this
    - Remember to #include <stdlib.h>!

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int num;
    num = atoi(argv[1]);
    printf("%d\n", num);
    return 0;
}
```

Numbers instead of Strings

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int sum = 0;
    for (int i = 1; i < argc; i++) {
        printf("The command line arguemnt at argv[%d] is %d.\n", i, atoi(argv[i]));
        sum += atoi(argv[i]);
    }
    printf("The sum is %d.\n", sum);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
z3548950@vx16:~/public_html/COMP151126T2/code/week4$ ./prog 3 67 58
The command line arguemnt at argv[1] is 3.
The command line arguemnt at argv[2] is 67.
The command line arguemnt at argv[3] is 58.
The sum is 128.
```

Problem Time

- Read in two numbers from the command line arguments and state whether the two numbers are the same or not
- Read in two strings from the command line arguments and compare the strings to say whether they are the same or not

What We Learnt

- Recap 2D arrays
  - Using them practically
- Recap strings
  - fgets
  - fputs
- Command line arguments
  - Getting input from the user

Reach Out

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

Thank You

Questions?