COMP1511 - Programming Fundamentals

Term 3, 2019 - Lecture 9

What did we cover last week?

Arrays

- Two dimensional arrays

Functions

- Separating code for reusability and readability

CS Paint

- An explanation the first Assignment

What are we covering today?

Assignment 1

- Some details on Assessment

Functions and Libraries

- Including files and libraries

Characters and Strings

- A new variable type!
- Using letters and words in C

Recap - Arrays

Storage for sets of identical variables

- Declared at a specific size using square brackets **[ ]**
- A single name for the group of variables
- Individual elements of the array are accessed by index (an integer)

Two Dimensional Arrays

- We can declare arrays of arrays, which allows us to make grids of variables
- We usually use rows and columns to index them

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]);
    }
```

Assignment 1

From a practical perspective . . .

- You will write a C program called CS Paint
- It will all be in a single file called `paint.c`
- Submission is through the `give` system

A recap of the basics

CS Paint

- A 2D array as the canvas
- Your program will receive integers as standard input that are drawing commands
- You will interpret those commands and then make changes to the canvas
- CS Paint is already capable of writing the canvas to the terminal, so you don't need to worry about that

Sequence of Commands

Commands will always be in a particular sequence:

- First integer is the type of command
- Other integers are the extra information that command needs
- Your program will receive one or more commands
- You will process each command in turn
- It's reasonably easy to process the entire command before moving onto the next one (rather than trying to process them all at once)
- When the commands are all finished, you will then print the canvas once to standard output (with the function we've provided)

Submit early, submit often

Using “give” will record your submission and back up your work

- It’s much harder to lose your assignment code if we have it!
- If things go bad, you can roll back to previous versions
- You can access your previous versions using our git repository
- The following link is also available in the assignment page:

`https://gitlab.cse.unsw.edu.au/z5555555/19T3-comp1511-ass1_cs_paint/commits/master`

How will your code be tested?

Your program will be run with a series of test cases

- These tests will not be exactly the same as our autotests
- Remember to check all possible inputs you can think of
- Writing your own test files is potentially very useful

Marking

How do you earn marks in this assignment?

- **Close to a pass (40-50%)**
  - A solid attempt at stage one
  - Being able to draw some lines, but not all possible cases
  - Not necessarily dealing with multiple commands
- **Pass (50-64%)**
  - Code runs without errors
  - Able to draw vertical and horizontal lines
  - A serious attempt has been made at the assignment
  - A higher mark will be given for squares and dealing with multiple commands

Marking Continued

- **Credit (65-74%)**
  - Successfully implements all of Stage 1
  - Some effort on Stage 2 will push marks higher
  - Code is reasonably readable
  - Shows some use of functions
- **Distinction (75-84%)**
  - Successfully implements both Stage 1 and 2
  - Any effort on later stages will award more marks
  - Code is easy to understand and readable
  - Uses functions to separate code for readability

Marking Continued

- **High Distinction (85%+)**
  - Successfully implements Stages 1-3
  - Stage 4 completion will push marks closer to 100%
  - Code is perfectly explained and elegant to read
  - Functions are used extensively to organise code

Free Marks!!!

Yep . . . get them right here!

Make your code understandable and readable!

- Follow the Style Guide
- This means correct indentation and consistent use of bracketing
- Use variable names that are understandable to a reader
- Have clear comments explaining your intentions (even if the code is not functional)
- Structure your code file so that different sections are clear
- Use functions to separate repetitive code

Questions?

Feel free to ask any questions now!

- Help Sessions have been expanded for one on one consultation if you need help with problems
- There's now a Help Session on every day of the week
- Details are on the Course Website

Recap of Functions

Code outside of our main that we can use (and reuse)

- Has a name that we use to call it
- Has an output type and input parameters
- Has a body of code that runs when it is called
- Uses return to exit and give back its output

Functions in Code

```
// a function declaration
int add (int a, int 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

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
#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;
}
```

Break Time

CS Paint Hall of Fame

- If you're a fan of Challenge Exercises that award bonus Marcs (not in any way related to course marks)
- We have some advanced challenges that are not necessarily related to programming, but are related to CS Paint
- Check out the assignment specification on the course website to see more info

Characters

We’ve only used ints and doubles so far

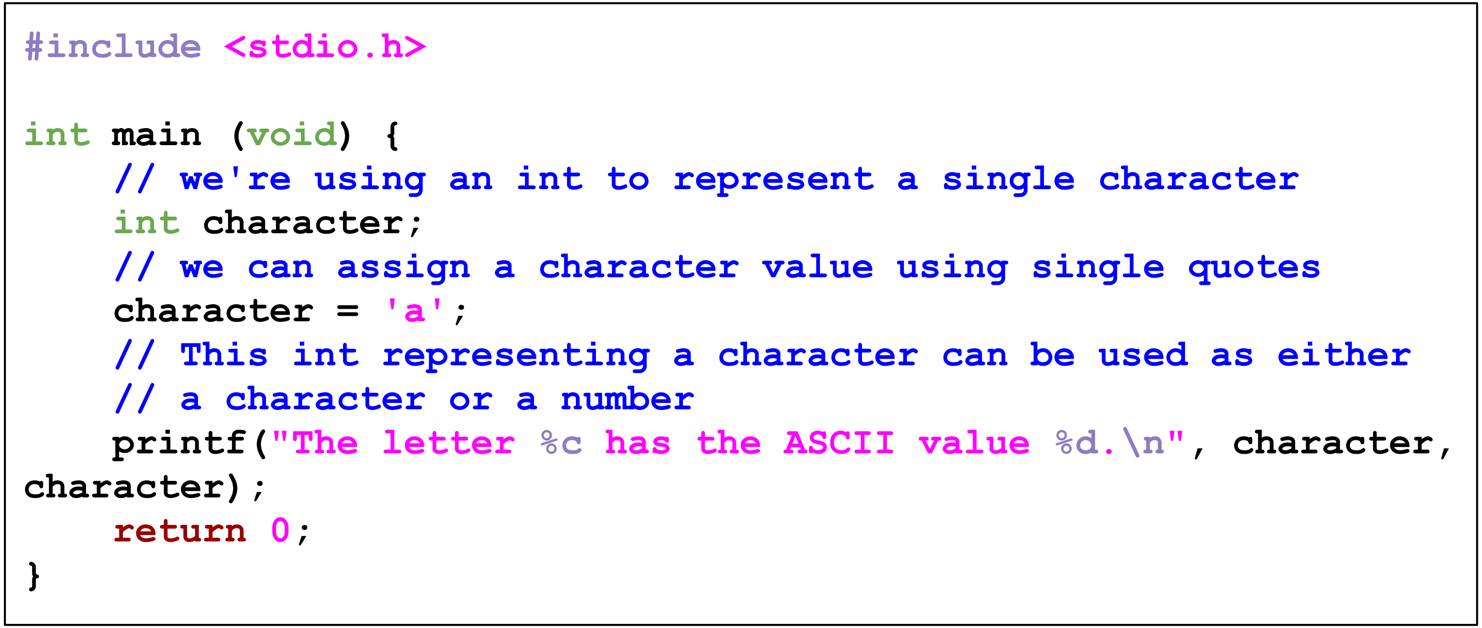
- We have a new type called **char**
- Characters are what we think of as letters, like ‘a’, ‘b’, ‘c’ etc
- They can also represent numbers, like ‘0’, ‘1’,’2’ etc
- They are actually **8 bit** integers!
- We use them as characters, but they’re actually encoded numbers
- ASCII (American Standard Code for Information Interchange)
- We will not be using **char** for individual characters, but we will in arrays

ASCII and Characters as numbers

We make use of ASCII, but we don't need to know it

- ASCII specifically uses values 0-127 and encodes:
  - Upper and Lower case English letters
  - Digits 0-9
  - Punctuation symbols
  - Space and Newline
  - And more . . .
- It's not necessary to memorise ASCII, rather it's important to remember that characters can be treated like numbers sometimes

Characters in code



```c
#include <stdio.h>

int main (void) {
    // we're using an int to represent a single character
    int character;
    // we can assign a character value using single quotes
    character = 'a';
    // This int representing a character can be used as either
    // a character or a number
    printf("The letter %c has the ASCII value %d.\n", character,
character);
    return 0;
}
```

Note the use of %c in the printf will format the variable as a character

Helpful Functions

`getchar()` is a function that will read a character from input

- Reads a byte from standard input
- Usually returns an int between 0 and 255 (ASCII code of the byte it read)
- Will sometimes return a -1 to signify end of input (which is why we use an int, not a char)
- Sometimes getchar won't get its input until a newline is entered

`putchar()` is a function that will write a character to output

- Will act very similarly to `printf("%c", character);`

Use of getchar() and putchar()

```
// using getchar() to read a single character from input
int inputChar;
printf("Please enter a character: ");
inputChar = getchar();
printf("The input %c has the ASCII value %d.\n", inputChar, inputChar);

// using putchar() to write a single character to output
putchar(inputChar);
```

Invisible Characters

There are other ASCII codes for “characters” that can’t be seen

- Newline(`\n`) is a character
- Space is a character
- There’s also a special character called EOF (End of File) that signifies that there’s no more input
- EOF has been #defined in stdio.h, so we use it like a constant
- We can signify the end of input in a Linux terminal by using Ctrl-D

Working with multiple characters

We can read in multiple characters (including space and newline)

This code is worth trying out . . . you get to see that space and newline have ASCII codes!

```
// reading multiple characters in a loop
int readChar;
readChar = getchar();
while (readChar != EOF) {
    printf("I read character: %c, with ASCII code: %d.\n",
            readChar, readChar);
    readChar = getchar();
}
```

More Character Functions

<ctype.h> is a useful library that works with characters

- `int isalpha(int c)` will say if the character is a letter
- `int isdigit(int c)` will say if it is a numeral
- `int islower(int c)` will say if a character is a lower case letter
- `int toUpper(int c)` will convert a character to upper case

- There are more! Look up **`ctype.h`** references or **`man`** pages for more information

Strings

When we have multiple characters together, we call it a string

- Strings in C are arrays of `char` variables containing ASCII code
- Strings are basically words, while chars are letters

- Strings have a helping element at the end, a `'\0'`
- It’s often called the 'null terminator' and it is an invisible character
- This helps us know if we’re at the end of the string

Strings in Code

Strings are arrays of type char, but they have a convenient shorthand

```
// a string is an array of characters
char word1[] = {'h','e','l','l','o'};
// but we also have a convenient shorthand
// that feels more like words
char word2[] = "hello";
```

Both of these strings will be created with 6 elements. The letters `h,e,l,l,o` and the null terminator `\0`

h	e	l	l	o	\0

Reading and writing strings

fgets(array[], length, stream) is a useful function for reading strings

- It will take up to **length** number of characters
- They will be written into the **array**
- The characters will be taken from a stream
- Our most commonly used stream is called **`stdin`**, “standard input”

- **`stdin`** is our user typing input into the terminal
- We also have **`stdout`** which is our stream to write to the terminal

Reading and writing strings in code

```
// reading and writing lines of text
char line[MAX_LINE_LENGTH];
while (fgets(line, MAX_LINE_LENGTH, stdin) != NULL) {
    fputs(line, stdout);
}
```

- **`fputs(array, stream)`** works very similarly to printf
- It will output the string stored in the array to the standard output

Helpful Functions in the String Library

<string.h> has access to some very useful functions

Note that char* s is equivalent to char s[]

- `int strlen(char* s)` - return the length of the string (not including \0)
- `strcpy` and `strncopy` - copy the contents of one string into another
- `strcat` and `strncat` - attach one string to the end of another
- `strcmp` and variations - compare two strings
- `strchr` and `strrchr` - find the first or last occurrence of a character
- And more . . .

What did we learn today?

Assignment 1

- A recap and assessment

Functions and Libraries

- Accessing C libraries and their functions

Characters and Strings

- Expanding our variables to letters and words