Programming Portfolio

Release Notes

Student: Andrew Doyle

Student Number: 12252388

For the MSc in Computer Science (Conversion) at University College Dublin

December 24th 2012

Release Notes

Introduction

The release notes serve to demonstrate what each program in the portfolio does, and how they do it. Completing this document also served to reinforce the author`s knowledge of C programming.

Operating System Used: Windows 7 Home Premium 64-bit
Text Editor: Notepad++
Compiler Used: MinGW (as suggested in Lab Document 1)

Program 1

helloworld.c

The traditional hello world program has been created which is the first program that new C programming students usually write.

Program 2

helloworldnewline.c

Expanding on Program 1, hello and world are put on separate lines, to demonstrate the ability of \n.

Program 3

add.c

Two int variables are created, the values of which are added and displayed on the command prompt screen.

Program 4

sizes.c

This program displays the size of various variable types, followed by the size of different arrays. This is completed using the *sizeof* built-in library function.

Program 5

colour.c

This program demonstrates the capability of scanning user input, storing it in an array, and outputting the input to the command prompt screen.

Program 6

colourfirstletter.c

The ability to access particular element of an array is the main purpose of this program. The user input is scanned and stored in the character array, however, when printing to the command prompt only the first character of the array is accessed.

Program 7

colouroverflow.c

This program serves to demonstrate how a program may crash if user input is written to memory locations that were not supposed to be written into. Step 7 demonstrates how a small discrepancy may not crash a program, however in larger programs crashing is more likely.

Program 8 (from program 8 onwards, files are simply named 8.c, 9.c, 10.c and so on)

The user is prompted to enter two numbers, which are added to each other, and also subtracted from one another.

Program 9

Similar to program 8, however both the user input and the results of the calculations are stored in an *int* array.

Program 10

If/Else statements are put into practice in program 10. The user is asked to input 3 numbers and the largest of them is printed to the command prompt screen. The largest number is calculated by way of an *if-condition* which tests whether each number is greater than the other two numbers. Both numbers are compared using the && operator, which is the first time this operator is used in the portfolio.

Program 11

The aim of this program is to alter the given code so that the user is given the opportunity to enter a character, whereas previously a warning message was communicated and the user did not have the chance to enter a character. Now the program takes in the user input of both the word and the character, and displays it to the command prompt screen. This is made possible by means of a while loop.

Program 12

A simple calculator is built, the user having the option to multiply, divide, add or subtract. Two possible solutions have been implemented. Both have different methods of determining the operator entered by the user. **12.c** uses if-else statements. **12b.c** uses a switch statement. The advantage of **12.c** is that the user can enter x or * and the numbers will be multiplied.

The advantage of **12b.c** is that the user has the option of exiting the program without carrying out any calculations. The calculator is also always active unless the user requests to exit the program. This saves the user the hassle of running the program any should they wish to carry out many calculations.

Program 13

Program 13.c builds on 12.c by allowing the user to quit the program by entering q. This is completed by means of the else-if statements nested inside an if-statement which itself is nested inside a do-while loop.

The do-while loop is executed if the operator is not q. The user is asked what operation they wish to be carried out. If the operation is not q then the user is asked to enter the numbers and the calculations are carried out. Calling **getchar()** after the else-if statements ensures that the loop repeats successfully. This program has the benefits of both 12.c and 12b.c.

An alternative method as shown on the Moodle site is included for information purposes. The c file has less code and may be easier to read for some.

Program 14

From program 14 onwards a header is included as per the portfolio specification.

This program serves as a demonstration of the fact that unsigned integers contain positive number only. One may think that subtracting 2 from 1 should result in '-1', but this exercise shows that this is not the case.

Program 15

The ASCII character set defines 128 characters (0 – 127 decimal). A char variable is created which serves as the condition for a while loop which is set to run 128 times. The programme keeps on running through the numbers. The loop will only execute once if the loop condition is set to loop 126 times or less.

Program 16

This program counts how many characters are in the word entered by the user. An int variable named count is declared and given the value of 0. A character array is also declared. The amount of characters in the word is found by initializing the counter in the first element of the character array, and incrementing the counter by one **while** the null character is not found. This is completed with a while loop.

Program 17

Program 17 completes the same task as program 16 except it uses a for loop, instead of a while loop.

Program 18

The program knows a secret word and the program loops until the user guesses the word correctly. A constant is declared to store the secret word and a char array to store the users guess. While the users guess and the secret word are not identical, the user is asked to guess the secret word.

If the words are identical the program exits and the user is praised. The **strcmp** function is used to complete this task. If incorrect words are guessed the program loops until the correct word is entered by the user.

Program 19

3 guesses are only allowed by the user to guess the secret word, which builds on the functionality of program 18. This is achieved by use of a for loop, which executes as long as SECRET and guess are not equal, and count is less than or equal to three.

Program 20

Program 20/20b is an exercise which demonstrates the differences between **scanf** and **fgets** in terms of dealing with the newline character. Program 20 prints one more blank line than program 21 because **fgets** processes the newline character when the user presses enter, and prints the blank line to the display screen on the command prompt. However **scanf** leaves the newline character in the user input buffer.

Program 21

The user is asked to enter a sentence. The input is scanned and the words are printed back to the command prompt screen in reverse order (each word reversed, not the whole sentence).

While Loop

The first step take is to isolate the end of the sentence entered. This is done by identifying the null character '\0'. A while loop is executed if the null character is NOT found.

If Statement

Within the while loop an if-else statement is carried out. The if-statement is carried out if both the space character and the newline character are not found. The space character will identify the end of a word, as will the newline character since this will be the last word the user has entered.

The counter **j** in the variable **word** is assigned the value of the counter **i** in the user **input** array. Both j and I are incremented so that **word** now stores the first word input by the user. Now, after looping through the

first word the space character ‘ ’ will be encountered so the loop no longer executes. The else statement is now carried out.

Else Statement

The else statement has a nested while loop. The loop is carried out if **J** is greater than or equal to zero. **J** will now be the space character between the words. Within the loop the value of **J** - 1 within **word** is printed. If the word entered by the user is woof, then the character f will be printed. **J** is decremented by one so that **J** is now at ‘f’ (taking the woof example again). The loop starts again and prints **J** - 1 (which will now be the character ‘o’). The loop continues until **J** reaches 0 i.e. the first letter in **word**.

After the while loop **J** is set to be zero again so that the if-statement can be carried out again successfully on the next word. **I** is incremented by one; this skips the space character and assumes the first character of the next word. The initial while loop is carried out again until the NULL character is found (end of sentence).

Program 22

Program 22 takes in user input and replaces the vowels in the words with the ‘$’ symbol. A char array named ‘sentence’ stores the user input. A counter **i** of type **int** is declared for use in a for-loop, and an int named **len** is declared which will represent the string length of the sentence later in the program.

The user input is stored in **sentence** and **len** is assigned the string length of **sentence**. A for loop is created with the condition that **i** (the counter) is less than **len**. The statement tests to see if **i** is a vowel, and if it is, it assigns **i** the ‘$’ symbol. The condition is now tested again and carried out if true. The vowels will have been replaced by the ‘$’ symbol, and **sentence** is printed out to the command prompt screen after the for loop.

Program 23

Program 19 is altered so that a function is used to read the user`s input. Instead of calling scanf in the main, a function is used to carry out the same task. A function prototype is declared before the main, and is defined after the main to read the users input. It is then called within the main and it scans the user input as defined by the function.

Program 24

Program 24 provides the same functionality as program 21 whilst using a function as in program 23.

Program 25

The occurrence of each alphabetical character in the users input is counted. A for loop is executed if **I** is less than 26 (**I** represents the element in the letters array). A while loop is executed within this for loop when **j** is not the newline character (**j** represents the element in the user input). The statement in the while loop tests if the element in **I** is equal to the element in **j**, and if it is, **count** is incremented by one. After this **j** is incremented by one and the while loop is tested again.

The count of the letter is printed to the command screen. **J** and **count** are set to zero and the for-loop is executed once more, until all the letters have been tested.

Program 26

The purpose of this program is the test the functionality of the second main, which is used to read user input on the command line.

- 26.c tests the program as per the lecture notes. The program tells you have many arguments you entered in the command line and prints them out.
- 26b.c sets **i** to be 5. As before it tells you how many arguments you entered, but will only print arguments entered from argument 5 onwards.
- 26c.c was carried out as a test by the author to see if entering 10 in argv[] would make any difference, alas it does not.

Program 27

Program 27 demonstrates the ability to allocate memory using malloc. **Mem** is allocated 50 bytes of memory using a **const** called **bytes** which was declared before the main. An if-else statement prints out the memory address and contents of the character array called **mem**; provided enough memory is available. Otherwise a warning message is displayed telling the user that insufficient memory is available, and the program will then exit.

Program 28

Program 28 is an alteration of program 27 whereby a character array is used instead of malloc. This demonstrates how pointers and arrays are the same thing. Char *mem as in program 27 is the same as char input[] as used in program 28.

Program 29

A simple calculator program whereby the user specifies the numbers and the operator on the command line is implemented. The second main is used to allow command line input whilst **sscanf** is used to convert the character array to an integer. **Sscanf** scans the user input and stores it either numbers[0], numbers[1], or the operator. The calculations are implemented as per Program 13.

Program 30

Program 30 demonstrates how one can access and modify elements of an array using pointers. The user enters words which are store in the character array called **word**. The pointer **ptr** is made to point to the second character in **word.** New values are assigned to ptr elements. Printing **word** will show how it has been changed to reflect the program assigning new values using the **ptr** elements.

Modifying the second element in **ptr** to be a '_' actually results in a change to the third element of the original user input; since **ptr** was made to point to the second character in **word**.

Program 31

Building on program 19, the secret word is read from a text file instead of being declared in the program. A file pointer is declared and **fopen** is used to open the file containing the word to read. A check is carried out to make sure the file is opened successfully; if not, a warning message is display. **Fscanf** is used to scan the text file and store what it scans in **SECRET.** The file is then closed. The rest of the program is then the same as program 19.

Program 32

A text file was copied as a result of completing this program. The files are specified within the program source code. A better solution would be for the user to specify the files to be opened and copied to.

Program 33

Building on program 31, there are now 20 words in the text file. The user must guess each word consecutively. For ease of testing, the words are monday through sunday, january through december, and finally weekend.

As in program 31, the file is opened and read. A char variable named **ans** is declared to be a two dimensional array to allow the storage of twenty lines ten characters long. Counters are declared which are used in the loops and statements described below.

The first character in the file is read, and so long the End-of-File is not reached, **Fscanf** is used to scan 10 characters and store what it scans in the constant **SECRET**. **Strcpy** is used to copy the value of **SECRET** into the first element of **ans**. Int **i** represents the element in **ans**. **I** is then incremented so that on the next iteration of the loop the next word will be store in the next element of **ans. ch** is then called at the end of the loop and the loop iterates again, until the end of file is reached and all the words from the file have been store in **ans**. The file is then closed.

A **for-loop** is put in place, z being used to represent each word; the loop is executed 20 times. Next, another **for-loop** is nested with the condition that **count** is less than 3. This is to ensure there are only 3 guesses allowed. Unusually there is no statement in the for-loop brackets; the demonstrator suggested this was the best way of implementing the solution for this part of the program. Within this **for-loop** the user is asked to guess the secret word with their input being stored in **guess**, and an **if-else statement** is then carried out to determine whether the guess was correct.

Guess is compared to **ans** and if they are equal, the user is congratulated and the **break** keyword is used to exit the loop, meaning the user will be asked to guess the next word. Otherwise **count** is incremented by one and the current for-loop is iterated again.

At the end of the outer for-loop, a condition is set to say that if **count** is greater than or equal to three, the program will **break** i.e. exit.

Program 34

This program works by typing *34 test.txt result.txt* in the command line prompt.

Two text files are compared for program 34. As in previous programs, the second main along with sscanf is used to read user input on the command line. The files are opened. **Ca** and **cb** are given the value of the first character in each file. A while loop is executed if the end-of-file is not reached and **ca** and **cb** match. The loop assigns **ca** and **cb** the next character in their respective files. The loop will end when the end of file is reach or when **ca** and **cb** do not match.

If **ca** and **cb** match at this stage the user is told that the files are identical, otherwise they are told the files differ. Both the text files are subsequently closed.

Program 35

This program works by typing *35 lines.txt* in the command line prompt.

A file with 80 lines is opened to read; the aim of this program is to display 20 lines at a time to the user, who has the option of quitting of displaying the next 20 lines. The first character in the file is read and while it is not at the end of the file it is displayed to the screen using **putchar(c).** If the character is the newline character **linecount** is incremented by one.

When linecount reaches 20, linecount is reset to 0, and the user is asked to either quit (**break** keyword) or display 20 more lines (by reading the next character from the file and iterating the while-loop again).

Program 36

Program 36 counts the amount of brackets in the file and tells the user whether the amount of opposing brackets are equal or not. Counters are initialized for the 6 brackets. The file is opened and the first character is read. If the character found is one of the bracket types, the relevant character counter is incremented. The next counter is read and the loop iterates until the end of the file.

The opposing brackets are compared and if they are equal the user is told the amount of opposing brackets are identical otherwise the user is told the count does not match. For convenience the amount of each bracket is then printed at the end of the file.

Program 37

Program 37 is the first use of structs in the portfolio. The user is able to enter student records and write them to a text file, or read the contents of the file. Three functions are declared, one to enter information, one to write to the file, and one to read from the file.

A structure is created to store the students name, phone number and student number. A structure called **andrewdoyle** is called in the main function block. The input function takes the user input and stores it in the relevant member variable of the struct. The write function opens the file students.txt to write (if it does not exist it will create the file). It will write what has been entered by the user to the file using **fprintf**.

The read function will open the file to read, scan each member variable of the structure and print it out to the command prompt screen. The user is given a menu selection built using the **switch case** method. The functions are called in the main; they are implemented when the user selects their option.

Program 38

Program 38 expands on program 37 by allowing the user to process more than one student. An array of structs lets the user do this. In the input and write functions, the syntax is slightly different, for example, &andrewdoyle->firstname is now written as andrewdoyle[i].firstname. An int is declared to allow access to the array of structs.

Furthermore, in the read structure, a for-loop is created to specify how many records are read on the command prompt screen. The rest of the program code is similar to program 37.

One thing to note is that the code is currently set to read 5 structs to the command prompt screen. If you are editing the text file, and delete records so that less than 5 structs are in the file, gibberish will be displayed on the command prompt screen when reading the contents of the file.

Program 39

The author has attempted to alter program 38 to use a linked list unsuccessfully. As part of the learning process the author completed a basic link list which is included within its own folder in the 39 folder. Applying the knowledge from this exercise to program 39 did not work, and as such, the attempt completed on 39 thus far should be considered a work in progress.