CSCI 247 Computer Systems I

Laboratory Exercise 1

Objectives

1) Gain familiarity with using an editor, compiler and debugger
2) Gain familiarity with the C programming language constructs and keywords
3) Understand basic C data types and their respective sizes
4) Gain familiarity with string manipulation in C

Submitting Your Work

Submit your C program files as the Lab Exercise 1 Submission item on the course web site. You must submit your program by 11:59pm on the Friday following your scheduled lab session.

Lab Tasks

Task 1: Learn how to create a C source file and compile and link it to get an executable

- Create a folder called "CSCI247" in your home directory
- Create a subfolder called "Labs" under your previously created "CSCI247" folder
- Create a subfolder called "Lab1" under your previously created "Labs" folder
- Use the "emacs" editor (or any editor of you choice) to create a file called "Lab1.c", type the following contents and save the file.

```
#include <stdio.h>

int main()
{
        printf("Wow! This is my first C Program!\r\n");
}
```

- Compile this file into an executable using the command line "gcc –o Lab1 Lab1.c"
- Run the executable ("./Lab1") and confirm you see the string "Wow! This is my first C Program!"

Task 2: Understand function calls

- Modify the "Lab1.c" to add a function called DataTypeSizes as follows:

```
/*
        Course: CSCI 247  - Fall 2017
        File Name: Lab1.c
*/

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

#define CHAR_DATA_TYPE      "char"

void DataTypeSizes(void);

int main()
{
        DataTypeSizes();
        return(0);
}

void DataTypeSizes(void)
{
        printf("\"%s\" is a standard C datatype. Size of a \"%s\" data type is = %ld\r\n",
                                        CHAR_DATA_TYPE , CHAR_DATA_TYPE, sizeof(char));
}
```

Task 3: Understanding key syntactical concepts of the C language

Study the changes you made above and identify where and why the following concepts are used (add comments at the top of the source file)

- Preprocessor directives – explain what is really happening behind the scenes.
- Forward declarations - Explain why it is required.
- Macros – Explain some of the advantages of using Macros.
- Functions – Explain why a function call is more advantageous than adding code to the main routine.
- Escape character(\) – Explain why an escape character is needed in a string for some characters
- Operators – When is the value evaluated for this operator? How is an operator different from a Macro or Function?
- Keywords – What is a Keyword and what are some of the keywords in this file?
- The printf function prints to the display. What is the significance of the display in the context of your program? (Hint: In Unix based systems all I/O APIs are directed at file based devices. There are 3 file based handles available to a program – stdin, stdout and stderr)

Task 4: Understand the basic C Data types

Modify the "DataTypeSizes" function to display the sizes of the other basic C type specifiers (int, float and double). These data types also allow for modifiers such as "signed", "unsigned", "short" and "long". Experiment with these modifiers and see if it affects the size.

Task 5: Understanding Arrays and passing arguments by reference

The C language does not have a "string" type. Instead a string is created by allocating memory for a contiguous sequence of characters. The end of a string is identified by a unique character known as the NULL character. You tell the compiler to allocate contiguous memory when you declare an "array" of any basic C type.

In your main function declare an array called "str" of 100 characters. Then add a function that uses the "**fgets**" function to read a string from stdin. Call this function from your main program and pass the starting address of the array to this function so it has access to the memory location where you array is located. The function declaration should be as follows:

```
char* GetStringFromStdin( char* str);
```

To prove you have read correctly, write another function to display the same string to the stdout using the "**fputs**" function and call that function from your main function after you have called the earlier function. The function declaration should be as follows:

```
void DisplayString( const char* str);
```

Understand the concepts behind passing a variable by reference or by value to a function in the above calls. Note the significance of the "const" qualifier in the above function.

Task 6: Passing arguments to an executable

A few interesting things happen when you start a program. All operating systems will have some form of a loader that loads a program into memory and create a stack for it. The stack is a special kind of memory that is used to store variables (function arguments) and return addresses when function calls are made. In C, your "main" program is conceptually not too different from functions you define and call from main and so the loader can actually pass arguments into your main program. Since all loaders need to follow some standard on what they can pass into the main function, C defines the following function declaration for the "main" function:

```
main(int argc, char* argv[]);
```

The first argument is the count of the number of parameters being passed in and the second argument is an array of character pointers that point to each of the strings in the command line, including the first string that has the name of your program.

If you declare your main function to not have the argc and argv parameters, you are simply ignoring the values populated on the stack for you by the loader.

Modify your Lab1.c file to include the argc and argv parameters and print their values at the very beginning.